99

Report to Landmark Graphics Corporation
University Partnership Program

Submitted By

Paul L. Stoffa

The University of Texas at Austin
Department of Geological Sciences and Institute for Geophysics
8701 Mopac Boulevard
Austin, Texas 78759-8345

Telephone: (512) 471-0465

University of Texas Institute for Geophysics Technical Report No. 99

Report to Landmark Graphics Corporation
University Partnership Program

Submitted by: Paul L. Stoffa,The University of Texas at Austin, Department of Geological Sciences and Institute for Geophysics, 8701 Mopac Boulevard, Austin, Texas 78759-8345, (512) 471-0465.

Interactive τ-p Velocity Analysis on the Landmark Workstation - Methodology

Paul L. Stoffa[†], Mark Wiederspahn[‡], Don Dean[‡], and Warren T. Wood[†]

ABSTRACT

Seismic processing requires accurate knowledge of the earth's velocity structure to properly image and interpret multi-fold seismic data. Conventional analysis methods are based on determining the best fit hyperbolas to seismic travel-times, T, as a function of source-receiver offset, X, in CMP gathers. The results of this analysis are the two-way normal-time, and the stacking velocity for each event analyzed. If the source-receiver offsets are not too large compared to the reflector depth, the stacking velocities can be equated to the RMS velocity. From knowledge of the RMS velocity and two-way normal times above and below a zone of interest, the interval velocity can be determined. Even if the earth is truly one-dimensional, i.e., velocity varies only as a function of depth, errors arise from the departure of the actual travel-times trajectories from the assumed T(X) hyperbola and the departure of the stacking velocity from the RMS velocity. These errors are in addition to the uncertainties involved in determining both the stacking velocity and the two-way normal-times from limited offset, band limited data in the presence of coherent and random noise.

An alternative interval velocity analysis method can be implemented if we first perform a plane wave decomposition of the seismic data. By transforming the data to the the domain of intercept time, τ, and horizontal ray parameter, p, velocity analyses can be performed exactly for a one-dimensional earth model without the need for intermediate quantities such as the stacking and RMS velocities. Workstation technology, such as the Landmark™, can then be used to do this velocity analysis interactively. For example, the original seismic data are plane wave decomposed on a remote computer, e.g., a Cray, and are then transferred either via ethernet or tape to the Landmark for interpretation. The interpretation is done directly in the τ-p domain by interactively

defining τ-p travel time curves and superimposing these curves on the τ-p data. Once reasonable agreement is achieved, the plane wave data are NMO corrected in the τ-p domain and then redisplayed. (The NMO corrections can be to two-way time or to depth.) The interpretation procedure is now repeated in the NMO domain to refine the velocity depth structure. The data can be windowed in time and ray parameter prior to analysis and the window changed during the interpretation process. The parameters determined directly by the interpreter are the interval velocity and the thickness and/or two-way normal-time of each layer. No approximations are required and all source receiver offsets are implicitly included in the analysis.

---

†University of Texas at Austin, Department of Geological Sciences and Institute for Geophysics, 8701 Mopac Boulevard, Austin, Texas 78759-8345

‡University of Texas at Austin, Instititue for Geophysics, 8701 Mopac Boulevard, Austin, Texas 78759-8345

INTRODUCTION

The τ-p domain has been used in various ways to filter and interpret seismic data. Stoffa et al. (1981), exploited the properties of the τ-p domain for wide angle reflection and refraction data to derive interval velocities. Tatham et al. (1984) and Tatham (1984) used the τ-p domain to separate seismic arrivals, e.g., reflections and ground roll, and compressional and shear events. The removal of multiples was also successfully performed in the τ-p domain by Brysk et al. (1986). Treitel et al. (1981) used the τ-p domain (actually angle of incidence) to image multi-fold reflection data. Recently, Pan et al. (1988) exploited the τ-p domain for the direct inversion of seismic waveform data.

In most cases, the τ-p domain has been used to either filter seismic data or as a way of deriving seismic interval velocities. Depending on the application, it may not be necessary to preserve seismic waveform amplitude and phase relations and the τ-p transform can be done by simple slant stacking, (Stoffa et al., 1981). In other cases, when the correct amplitude and phase are required, the cylindrical slant stack described by Brysk and McCowan (1986) should be used. Recently Wang and McCowan (1989), have shown that for a 1D earth the proper plane wave decomposition implicitly corrects for spherical spreading. Consequently, true amplitude processing is easily accomplished in the τ-p domain if the original data are adequately sampled in the offset domain.

In this paper we review the problem of seismic velocity analysis for a 1D earth structure in the τ-p domain. The analysis methods described are based on a reflection model of the subsurface not a diffraction model. This makes it possible to easily solve the velocity analysis problem in terms of reflection vertical delay times. The τ-p domain serves as a convenient domain for this analysis, because the problem can be formulated in a way that the required ray tracing is quickly

accomplished. This makes it possible to implement the methods described in a workstation environment.

τ–p TRAVEL-TIMES

The vertical delay time, τ, can be constructed geometrically as the time intercept at zero offset of a tangent to a seismic travel time trajectory in the source-receiver offset domain. Diebold and Stoffa (1981), and more recently Diebold (1987), developed the vertical delay time equations for planar dipping layers in 2D and 3D respectively. For a fixed source (or receiver) at position A on the surface of the earth and the receiver (or source) at position B, the 3D travel time equation is:

$$T_n = \vec{p}_b \bullet \vec{X}_n + \tau_n(\vec{p}_b) \quad , \tag{1}$$

where $\vec{p}_b = (p_{b_x}, p_{b_y})$, $\vec{X} = (x,y)$ and $\tau_n(\vec{p}_b)$ is the total vertical delay time.

For a 2D earth model, we can ignore the y dependance and write the vertical delay time contribution as

$$\tau_n(p_b) = \sum_{j=1}^{n} \Delta z_j \, (q_{a_j} + q_{b_j}) \quad . \tag{2}$$

where $q_{a_j} = \cos a_j / v_j$ and $q_{b_j} = \cos b_j / v_j$ are vertical slowness for the downgoing and upgoing ray paths, a_j and b_j are the angles of the down and upgoing rays with respect to the vertical, and $p_b = p_{b_x}$.

For a 1D earth structure the source vertical slowness contributions, q_{a_j}, and the receiver vertical slowness contributions, q_{b_j} are equal and equation 2 reduces to:

$$\tau_n(p) = \sum_{j=1}^{n} 2\Delta z_j q_j \quad . \tag{3}$$

where $q_j = q_{a_j} = q_{b_j}$.

The seismic velocity analysis method we will discuss is based on using the τ-p vertical delay time equation 3, to derive the thickness and interval velocity of each layer. The methods can be classified based on the seismic events used. For example, if only post critical reflection and refractions are used, the τ-sum method of Diebold and Stoffa (1981), can be used for a 1D earth structure. (An analog also exists for 2D and 3D earth structure, assuming the required up dip and down dip post critical reflections and refractions are observed.) Alternatively, pre- and post-critical reflections or just pre-critical reflections can be used. These are the methods described here.

In the analysis of reflection vertical delay times the methods can be either exact or approximate. The geometry used for the data acquisition, that is whether the data are common source/receiver or common mid-point, as well as the ray parameters used in the analysis, determine whether it is possible to use the approximate methods, or justify the use of the exact ones. For the reflection vertical delay times, the exact methods require that the overlying structure either be known explicitly or otherwise taken into account. That is, the vertical delay time differences between the reflection event being analyzed and the reflection event immediately above it are used to derive the interval velocity and thickness of the layer. In the approximate methods, only the total vertical delay times of the reflection being analyzed are used.

1D τ-p REFLECTION VELOCITY ANALYSIS

For 1D velocity analysis, we use equation 3. First, replace q_j by $(1 - p^2v_j^2)^{1/2}/v_j$ and then $2\Delta z_j /v_j$ by Δt_j, the two-way normal time in each layer:

$$\tau_n (p) = \sum_{j=1}^{n} \Delta t_j (1 - p^2 v_j^2)^{1/2} \tag{4}$$

To solve for the interval velocity and two way normal time we can measure the delay times of any two reflection events, e.g., τ_n and τ_{n-1} at the same two or more ray parameters. The differences between the vertical delay times, $\Delta\tau_n(p) = \tau_n(p) - \tau_{n-1}(p)$ as a function of ray parameter are used to solve for the interval velocity. For example, if we observe $\Delta\tau_n(p)$ for two ray parameters, e.g., p_k and p_l we can solve for the interval velocity, v_n:

$$v_j = \left[\frac{\Delta\tau^2(p_k) - \Delta\tau^2(p_l)}{\Delta\tau^2(p_k)p_l^2 - \Delta\tau^2(p_l)p_k^2} \right]^{1/2} \tag{5}$$

Alternatively, if we have many $\Delta\tau_n(p)$ measurements, we can do a linear least squares estimate of $\Delta\tau_n^2$ versus p^2. The intercept will be Δt_n, and from the slope, $v_n^2 / \Delta t_n^2$ we can recover the interval velocity, v_n.

Either of the above methods require that we first interpret the τ-p data to identify the delay time of the reflections, i.e., we must 'pick' the event times. This is usually possible, but may be difficult in practice. An alternative is to 'NMO' correct the τ-p data, (Stoffa et al., 1981). That is, given the correct interval velocity function, the n^{th} reflection event can be corrected to its total two way normal time. The τ-p normal moveout correction is:

$$\Delta T_n(p) = T0_n - \tau_n(p) \tag{6}$$

where $T0_n$ is the total two-way normal time,

$$T0_n = \sum_{j=1}^{n} \Delta t_j \quad .$$

Thus,

$$\Delta T_n(p) = \sum_{j=1}^{n} \Delta t_j (1 - (1 - p^2 v_j^2)^{1/2}) \quad . \tag{7}$$

To actually do the NMO, we resample the discrete τ-p seismic waveform data for each plane wave seismogram, $f(\tau,p)$:

$$F(T0_n,p) = f(\tau,p)\, \delta \left(\tau - \sum_{j=1}^{n} \Delta t_j (1 - p^2 v_j^2)^{1/2} \right) \quad , \tag{8}$$

interpolating as required for discretely sampled data. This process is repeated for all $T0_n$'s and ray parameters of interest to construct the τ-p NMO corrected data, $F(T0_n,p)$.

For a 1D earth, this process is equivalent to imaging the plane wave data directly to depth. Using equation 3, we note that

$$\Delta z_j = \Delta \tau_j(p) / 2q_j \tag{9}$$

and

$$Z_n = \sum_{j=1}^{n} \Delta z_j = \sum_{j=1}^{n} \Delta\tau_j(p)/2q_j \qquad (10)$$

The imaging to depth, or two-way normal time, can be done either in the τ-p domain by resampling the data as described by equation 8 or by phase shift followed by an integration over frequency in the ω, p domain. If the interval velocity function used for the imaging is correct, the reflections events for all the plane wave seismograms will be imaged to the same depth or two way normal time.

τ-p NMO is best implemented in a top down fashion. Once the velocity function is found for the section overlying the interval being analyzed, the τ-p data above this reflection can be NMO corrected and the next event's τ-p trajectory will be a single ellipse. At this point several methods can be used to define the residual τ-p delay times for the next event: In a workstation environment, τ-p travel time trajectories can be graphically superimposed on the data in an interactive fashion until visual agreement is reached (effectively 'picking' the next event delay times); the plane wave data can be iteratively τ-p NMO corrected until the event being analyzed is imaged to the same depth or two way normal time for all the plane wave seismograms; or, a two parameter semblance velocity analysis for interval two way normal time and interval velocity can be performed. For any of these approaches we have corrected for the overlying structure, i.e., we have 'layer stripped' or 'downward continued' to just above the layer being analyzed. Therefore, we have isolated the contribution of the current layer, and only a two parameter search is required

If the interval velocity of just one subsurface layer is of interest, it is not necessary to evaluate the entire overlying section. In this case, we can remove the effect of the overlying section by using an approximate velocity to correct the reflection event above the zone of interest. That is, we can do a single ellipse τ-p NMO (see below), followed by a residual statics correction,

if necessary. Alternatively, the reflection above the zone of interest can be 'picked' and the data static shifted. Another possibility is to window the data to include only a reference reflection from just above the zone of interest and the reflection from the base of the zone of interest. Then, compute the autocorrelation function for each windowed plane wave seismogram. The lag times will now follow an elliptical trajectory and can now be analyzed to determine the interval velocity of the zone using any of the methods described above.

In some cases, insufficient source-receiver offset coverage and hence a small range of observed ray parameters, may make the above exact analysis unnecessary. For small ray parameters a single ellipse can be used to approximate the true τ-p delay time (Stoffa et al., 1981). In this case, we expand the term $(1-p^2 v_j^2)^{1/2}$ and then normalize using the total two-way normal time:

$$\tau_n(p) = \sum_{j=1}^{n} \Delta t_j (1 - \frac{1}{2} p^2 v_j^2 + \ldots = T0_n \, (1 - \frac{1}{2} \, p^2 \sum_{j=1}^{n} \frac{\Delta t_j v_j^2}{T0_n} + \ldots \tag{11}$$

We recognize in the second term the RMS velocity,

$$\overline{V}_n^2 = \frac{\sum_{j=1}^{n} \Delta t_j v_j^2}{T0_n} \tag{12}$$

and the above expansion to second order is equivalent to the expansion for:

$$\tau_n(p) = T0_n \, (1 - p^2 \overline{V}_n^2)^{1/2} \tag{13}$$

By using the relations, $X = -\delta\tau/dp$, $\tau = T - pX$ and $p = dT/dX$, equation 13 can be shown to be exactly equivalent to the hyperbolic travel time equation:

$$T(X) = (T0_n^2 + X^2/\overline{V}_n^2)^{1/2} \quad . \tag{14}$$

Note that, in the above expansion only terms up to $p^2 v_j^2$ or sine squared were maintained. This indicates that when Snell's law ray bending increases to the point that contributions greater than the sine of the incidence angle squared become significant, the τ-p single ellipse delay time and the T(X) hyperbolic travel time approximation will fail.

EXAMPLE

We illustrate the τ-p NMO method for a 1D earth model, see Table 1. The synthetic data were generated directly in the τ-p domain by ray tracing, Figure 1. Figure 2 shows the data NMO corrected in the τ-p domain to two-way normal time. In this figure, the seismic events are horizontally aligned indicating that the velocity and thickness are correct. Since all ray parameter traces are corrected, this corresponds to a maximum reflection angle of 64° for layer one , 56° for layer two, and 25° for layer three. The stack over all the ray parameter traces is shown to the right of the NMO corrected data.

REFERENCES

Brysk, H. and D.W. McCowan, 1986, A slant-stack procedure for point-source data, *Geophysics*, 51, p. 1370-1386.

Diebold, J.B., 1987, Three-dimensional traveltime equation for dipping layers, *Geophysics*, 52, p. 1492-1500.

Diebold, J.B., and P.L. Stoffa, 1981, The traveltime equation, tau-p mapping, and inversion of common midpoint data, *Geophysics*, 46, p. 238-254.

Pan, G.S., R.A. Phinney, and R.I. Odom, 1988, Full-waveform inversion of plane-wave seismograms in stratified acoustic media: Theory and feasibility, *Geophysics*, 53, p. 21-31.

Stoffa, P.L., P. Buhl, J.B. Diebold, and F. Wenzel, 1981, Direct mapping of seismic data to the domain of intercept time and ray parameter: A plane-wave decomposition, *Geophysics*, 46, p. 255-267.

Tatham, R.H., and D.V. Goolsbee, 1984, Separation of S-wave and P-wave reflections offshore western Florida, *Geophysics*, 49, p. 493-508.

Tatham, R.H., 1984, Multidimensional filtering of seismic data, *Proc. IEEE*, 72, p. 1357-1369.

Treitel, S., P. Gutowski, and D. Wagner, 1982, Plane wave decomposition of seismograms, *Geophysics*, 47, 1375.

Wang, R.J., and D.W. McCowan, May 1989, Spherical Divergence correction for seismic reflection data using slant-stacks, *Geophysics*.

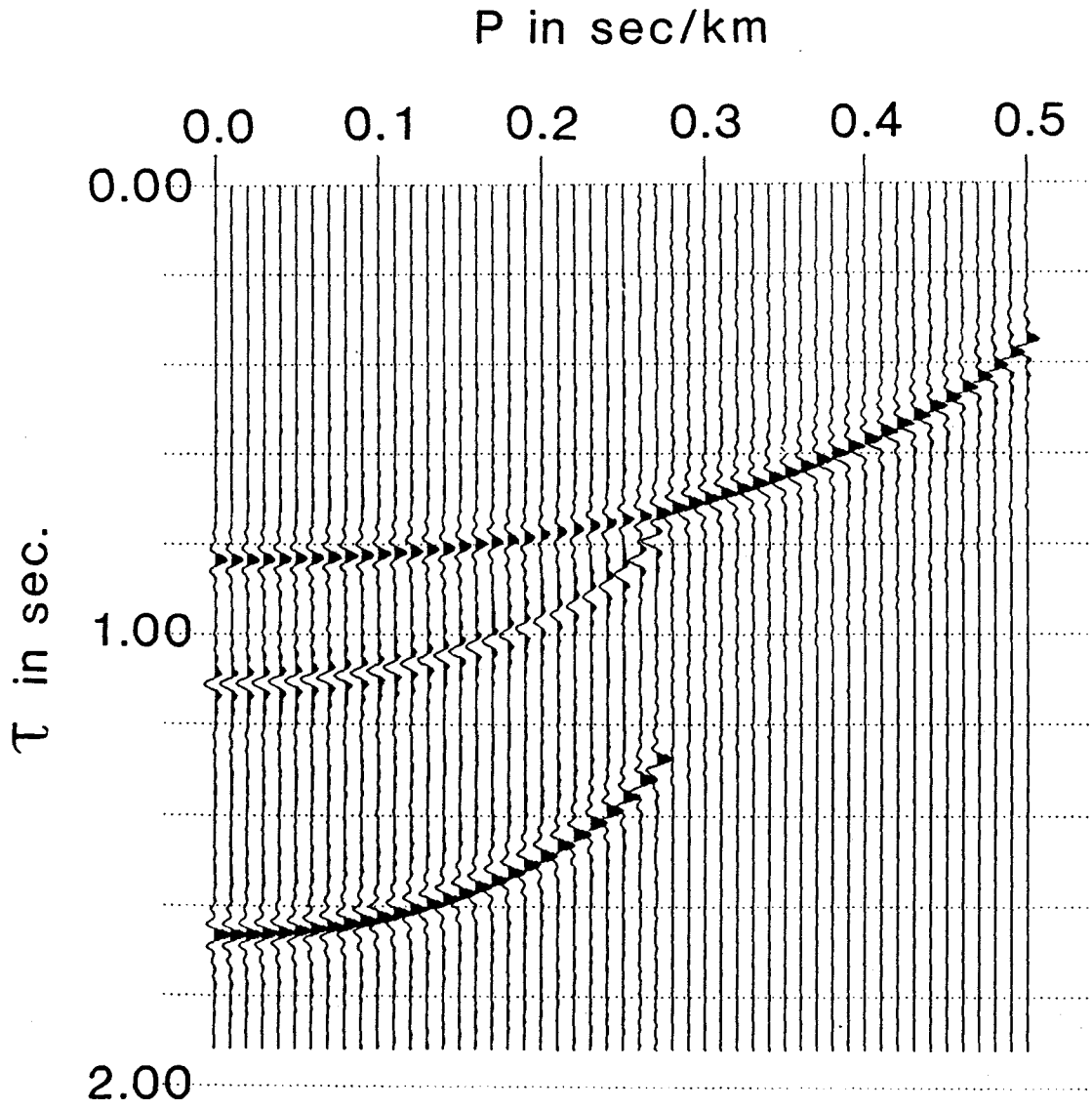


Figure 1. A synthetic τ-p shot gather obtained from a 1D earth model.

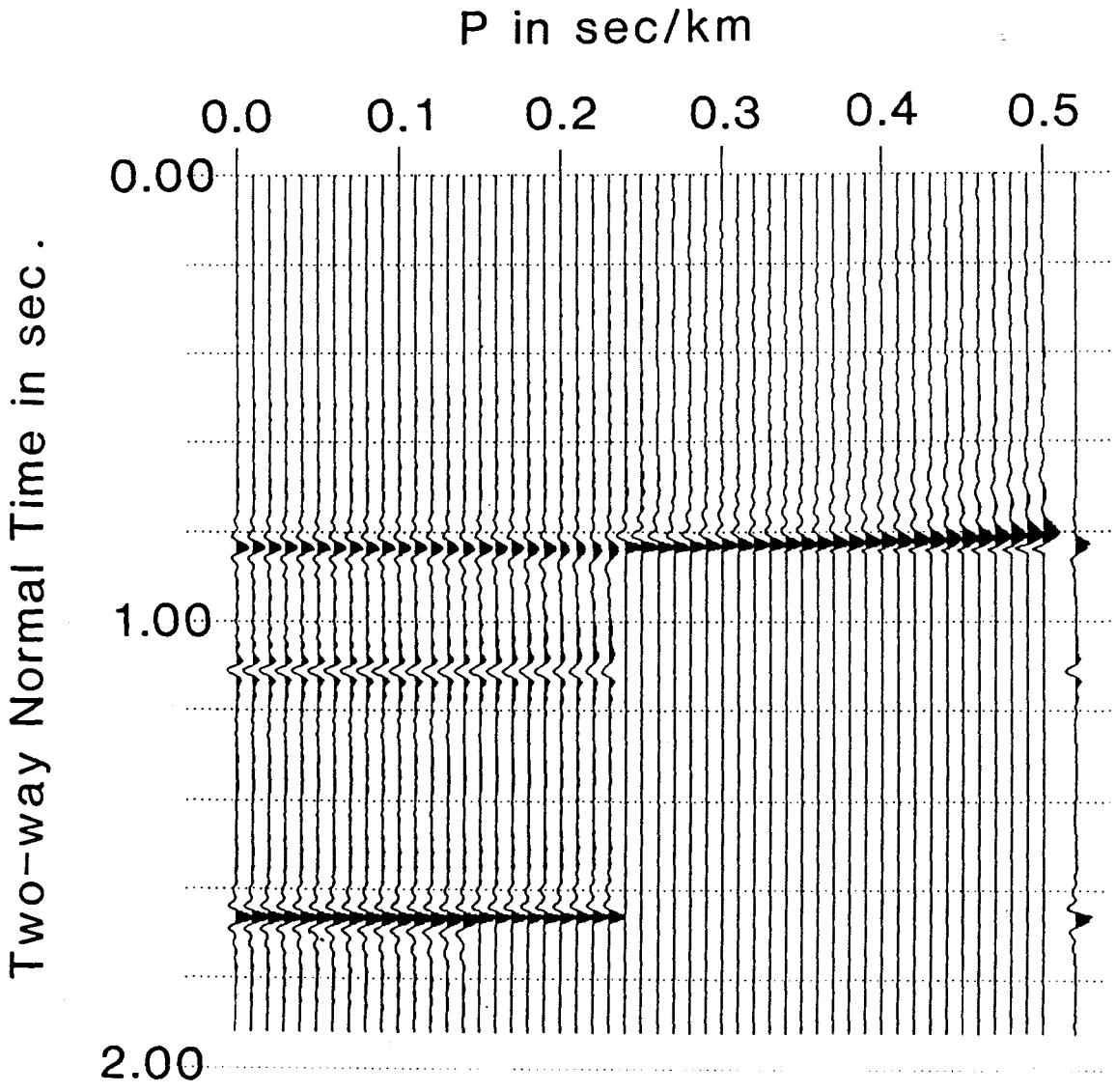


Figure 2. The same τ-p shot gather as in Figure 1 after the correct NMO is applied. The right most trace is the stacked trace.

Interactive τ-p Velocity Analysis on the Landmark Workstation - Examples

Warren T. Wood[†], Mark Wiederspahn[‡] and Paul L. Stoffa[†]

ABSTRACT

Five marine expanding spread profiles obtained from the Nankai Trough off Japan have undergone preliminary velocity analysis with the τ-p interval velocity analysis program which runs on the Landmark™ workstation. The program operates on data which has been transformed into the domain of τ and p and can perform an exact τ-p moveout analysis for a one-dimensional earth model. The velocity analysis consists of iterative forward τ-p travel-time modelling followed by a τ-p normal moveout correction. This method of analysis works well on the expanding spread profiles which have large source-receiver offsets, e.g., up to 20 km, and are common mid-point data. During the interpretation, several 'tricks' to speed up and refine the analysis procedure have been found. For example, since the moveout operation stretches the waveform at large p values, it is best to get a good fit of the τ-p travel-times to the data before applying the NMO corrections. Also, since the NMO correction is the most time consuming operation, the user may not wish to display all the available ray parameter traces on the screen until the final iterations. The interpretation of the expanding spread data are illustrated and the program will be available for the participants use during the meeting.

[†]University of Texas at Austin, Department of Geological Sciences and Institute for Geophysics, 8701 Mopac Boulevard, Austin, Texas 78759-8345
[‡]University of Texas at Austin, Instititue for Geophysics, 8701 Mopac Boulevard, Austin, Texas 78759-8345

DISCUSSION

The τ-p interactive velocity analysis program otherwise known as P-pick runs on the Landmark™ Workstation and operates on multichannel seismic data which has been transformed into the domain of intercept time τ and ray parameter p. The input to the program is a τ-p gather, and the output is a 1D earth model of layer thicknesses and velocities. The program is based on the exact 1D moveout analysis (Stoffa et al. 1981). The premise here is that when the reflection events in the τ-p gather are moved out correctly, the interval velocity model with which they were moved out is the correct earth model. This interpretation is done by superimposing τ-p travel time curves on the seismic data. As the model is adjusted the curves are redrawn until they match the seismic data. At this point the seismic data are moved out. If the reflection event in question is not completely horizontal the model is refined further.

The best way to see this process is with an example. This example comes from a long offset two ship experiment in the Nankai Trough off the southern coast of Japan. The largest offset in this experiment is 22 km and the largest incidence angles are ~65°. This profile is an expanding spread profile collected while a shooting ship and a receiving ship steamed toward a common mid-point (i.e., the same geographic position). The interval velocities we determine will apply to the sediments at the mid-point position.

Since these data were collected in deep water, the water bottom reflection is at $\tau_0 \sim 6.5$ sec. The reflection at $\tau_0 \sim 7.5$ sec is the oceanic crust. The velocities we are trying to determine are those of the sediments deposited on the crust. The first step is to display the data such that it fills the screen as in Figure 1. (In each of the figures the dotted timing lines are 200 msec apart). Since the water bottom reflection spans over 6 sec the time axis is very compressed and it is difficult to accurately fit a travel time curve to this reflection. We therefore have displayed only every 10th trace, enough to get an approximate fit. One very useful feature of this program is that the τ-p

travel time curves can be compared with the seismic data in the NMO domain as well as the τ-p domain. Once the majority of the NMO has been applied to a reflection, the time scale can be expanded and very fine adjustments to the model can be made. When this is done to the data in Figure 1 we can look at every trace of the data with an expanded time axis, e.g., as in Figure 2. This is accomplished by first picking the τ_0 time with mouse button 1 and then choosing an interval velocity with mouse button 3. (Although most menu options in this program are self explanatory there is a brief manual in the Appendix.)

There are several important things to note in Figure 2. Since the wavelet in this data has a small precursor which is not always visible, there is a question as to where the event should be picked. It was decided for the sake of consistency that all events would be picked at the zero crossing just prior to the main lobe. Therefore, the layer thicknesses will not be affected by whether or not the precursor is visible. Also important to note is that the post critical events, which occur at large values of p, may be phase shifted to such a degree that the user may try to fit the reflection by using a higher, incorrect velocity. Instead, one should initially concentrate on making only the precritical portion of the reflection data flat. The large amplitude post critical events will detract from the linear appearance of the reflection trajectories but it is important to remember that we are determining interval velocities, not necessarily the velocities which will make the best stack. Refinements can later be made to the model that include the post-critical events. The water bottom reflection in Figure 2 is a good example of a reasonably good NMO corrected reflection.

Once a reflection is horizontal the picking procedure is applied to the next reflection. We add a layer, pick the τ_0 time, adjust the velocity, re-window the data as needed, and apply the τ-p moveout. The curve and reflection event are re-examined in the NMO domain and if the precritical portion of the reflection is not flat, the velocity is adjusted and the data moved out again. Since adjusting the velocity of a layer will change the shape of all the curves below it, it is usually desirable to work from the top downward, only proceeding to the next reflection when the present

one is as flat as possible. In Figure 3 we see the result of this process after several iterations. The trace on the far right of the picture is a stack at the current state of moveout. Figure 4 is an example of how the data can be windowed to get a closer look. Any range of τ or p can be expanded to fill the screen, e.g., the data within the box in Figure 4 fills the screen in Figure 5. Here the final picks are made for this gather and the moved out version is shown in Figure 6. Figure 7 is the completely moved out gather and Table 1 is the final model file which shows the interval velocity and thickness for each layer. Figure 8 is a graph of time vs. interval velocity for this model.

TABLE 1. Interval Velocities and Interval Thicknesses of the Final Model

v_{int}	Δz
1.498	4.897
1.644	0.057
1.585	0.044
1.504	0.055
1.715	0.037
1.716	0.108
1.830	0.118
1.547	0.060
1.838	0.083
2.049	0.056
1.969	0.074
2.230	0.078
1.851	0.123
4.086	0.170

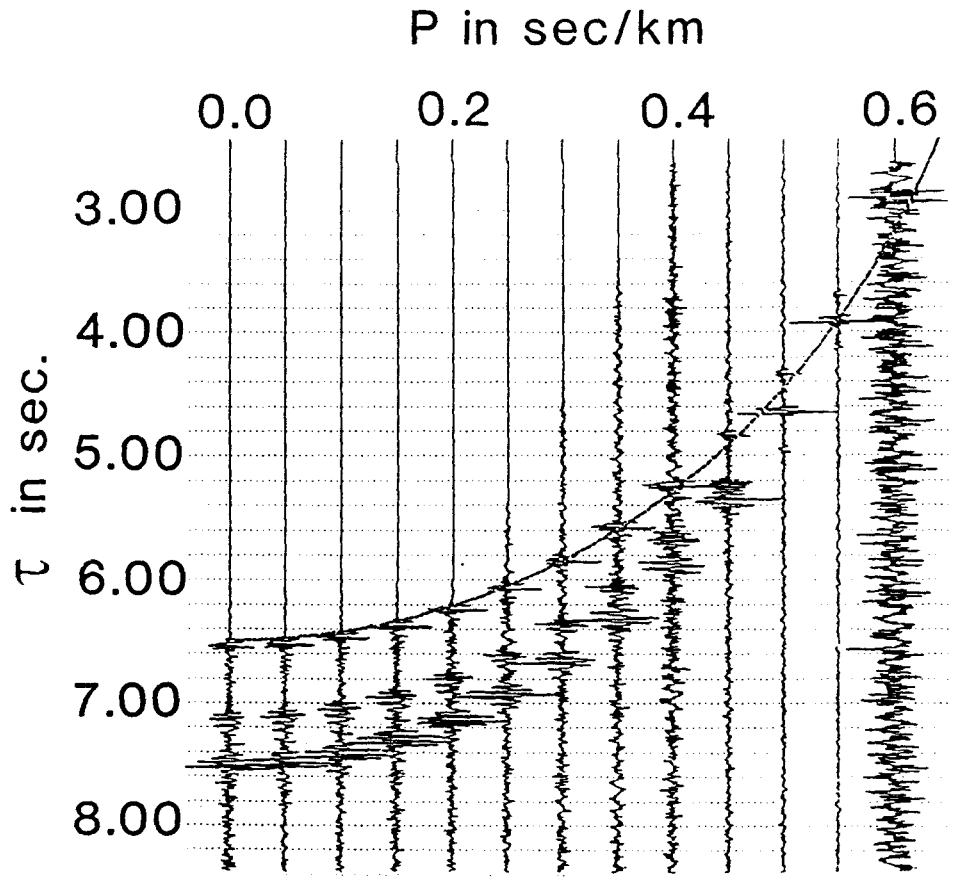


Figure 1. Every 10th trace of an ESP τ-p gather ready for interactive velocity analysis. The first model layer is drawn to get a rough fit to the water bottom.

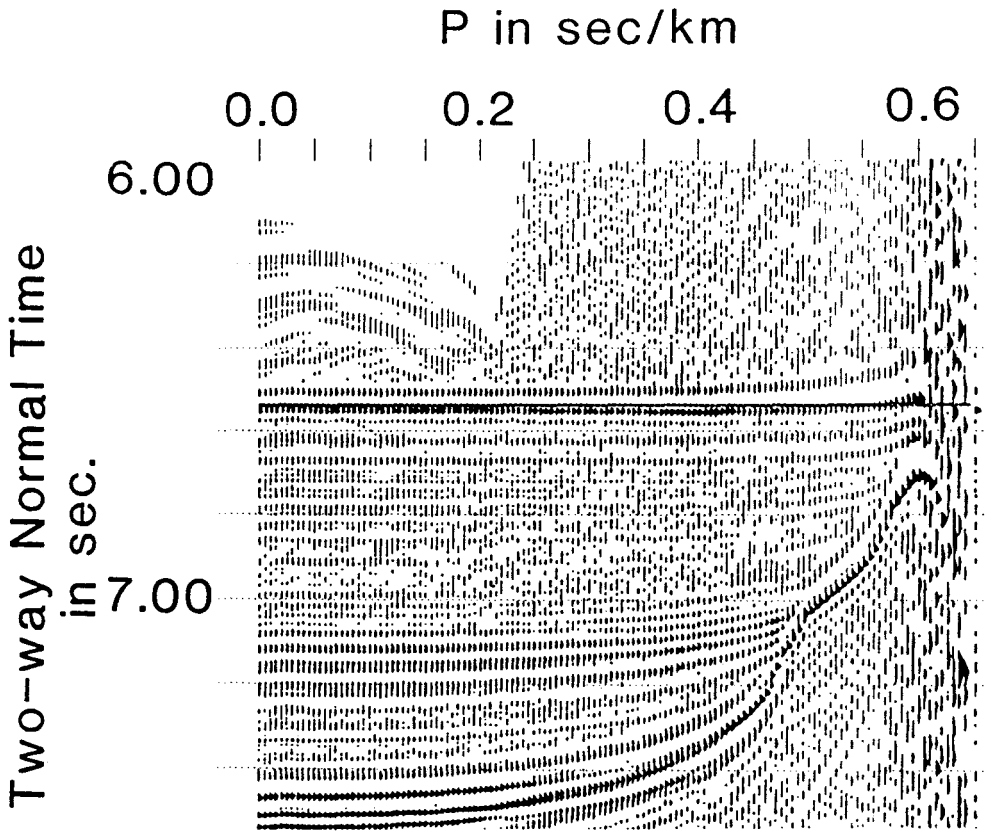


Figure 2. The same gather as in Figure 1 after the water bottom was moved out and every trace was displayed.

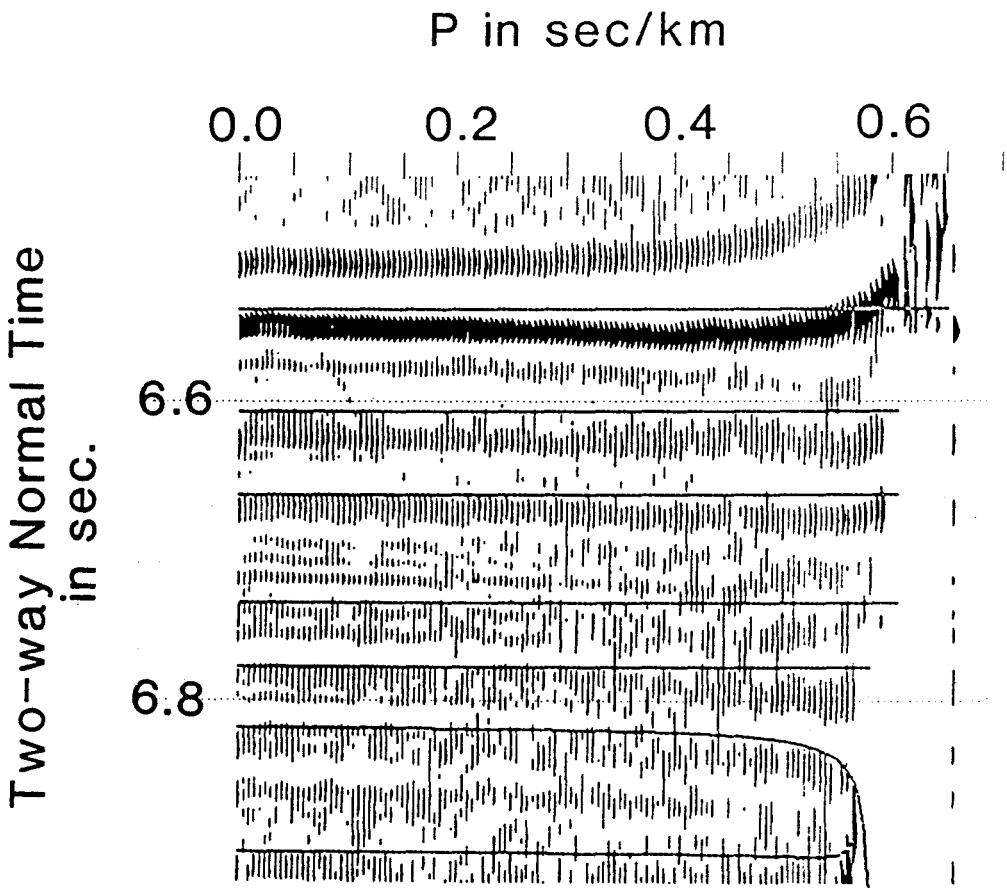


Figure 3. The upper 400 msec of the gather after several iterations of velocity picking and NMO.

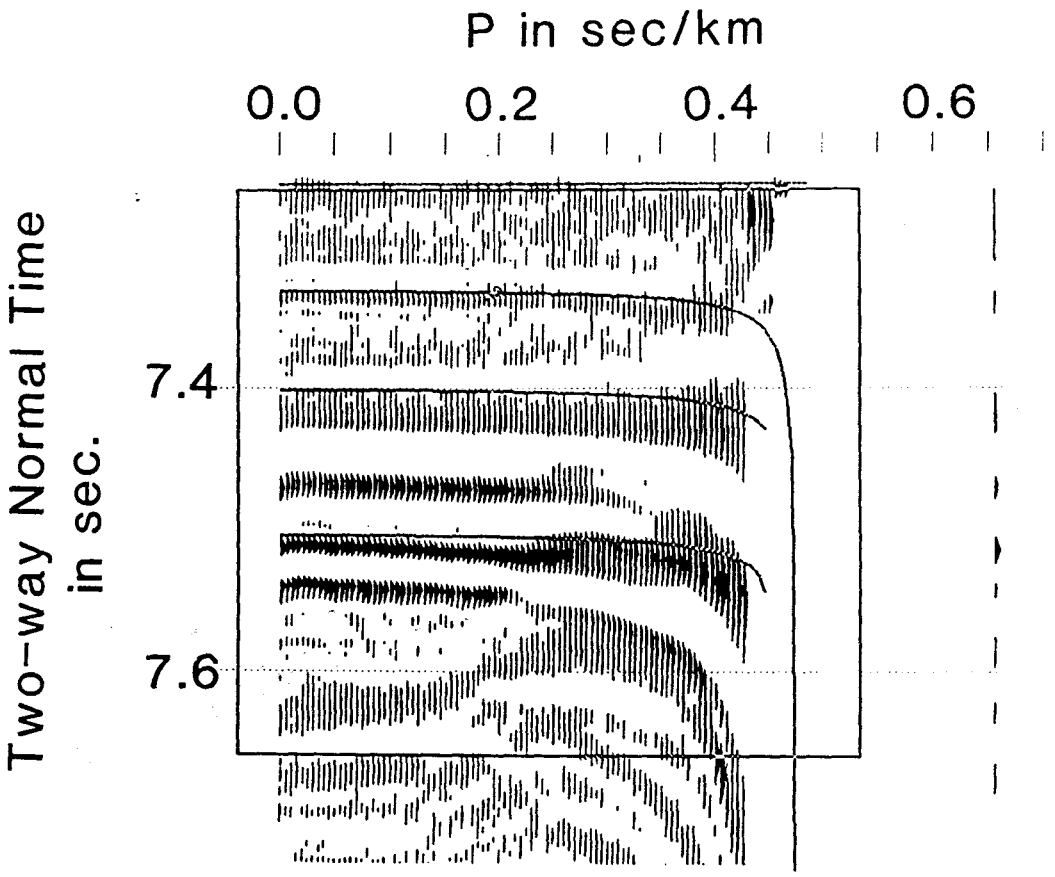


Figure 4. The bottom 400 msec of the gather being windowed for a closer look as shown in Figure 5.

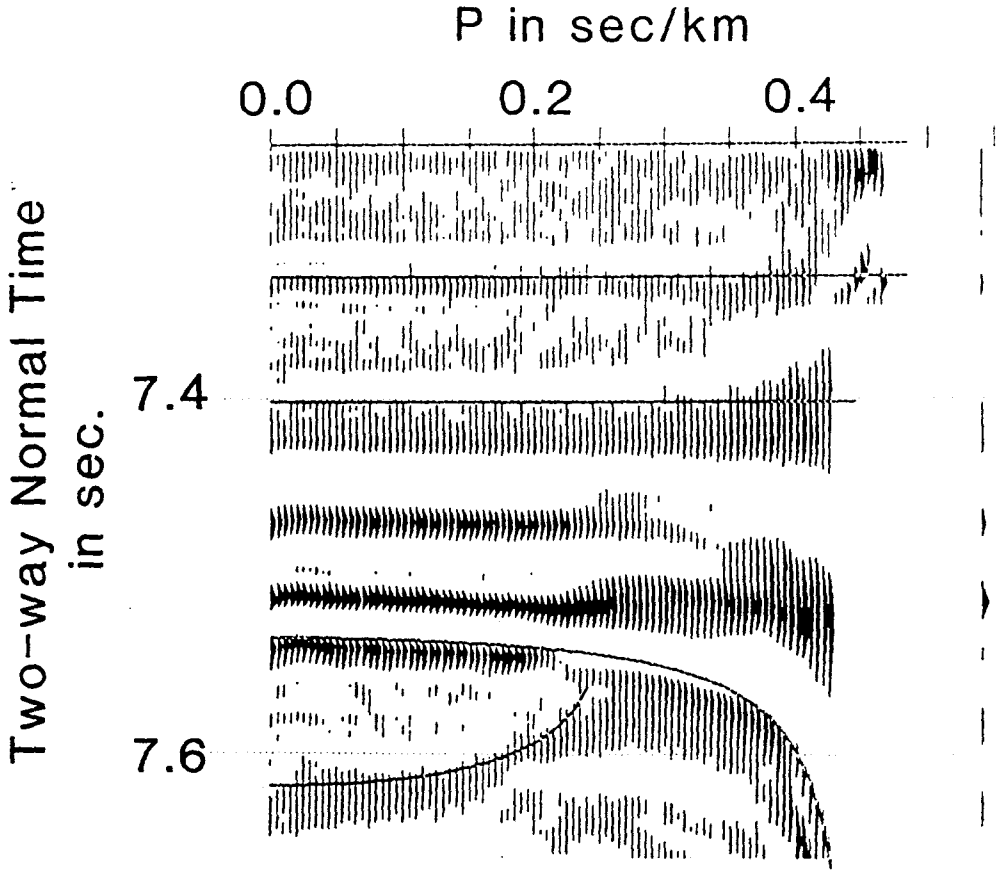


Figure 5. The last layers of the gather are picked and ready for NMO.

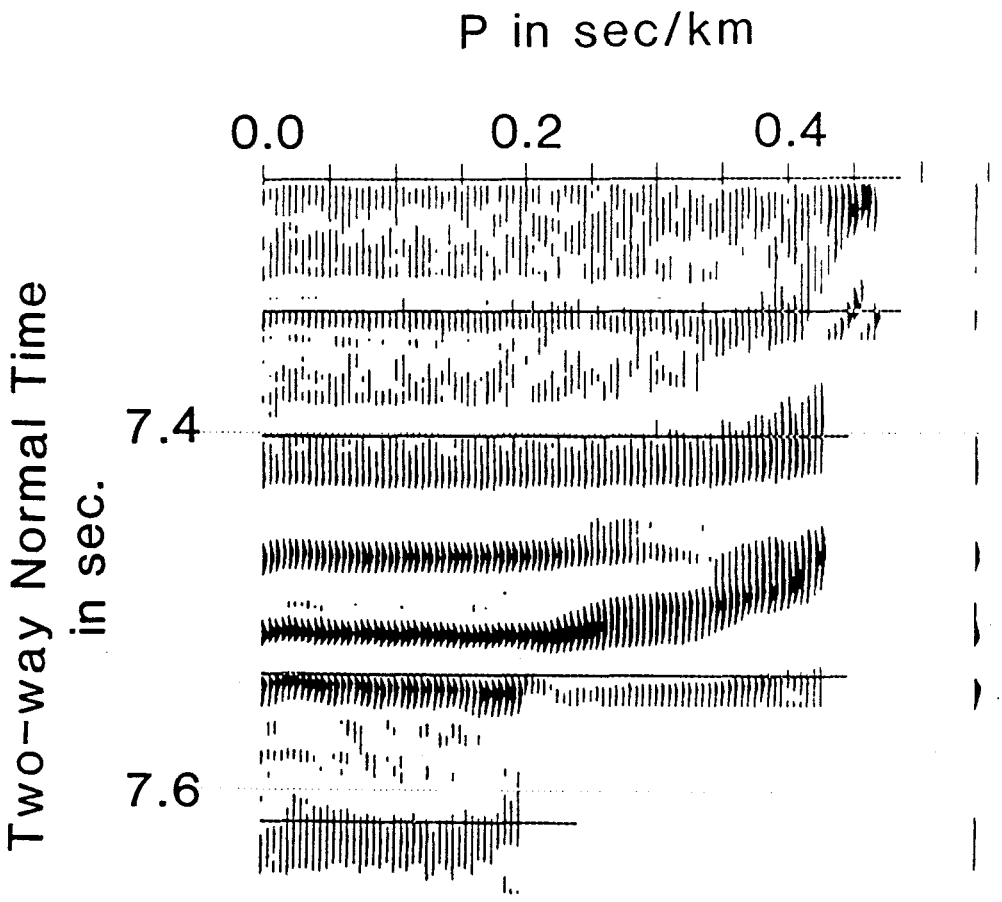


Figure 6. The same data as in Figure 5 after NMO.

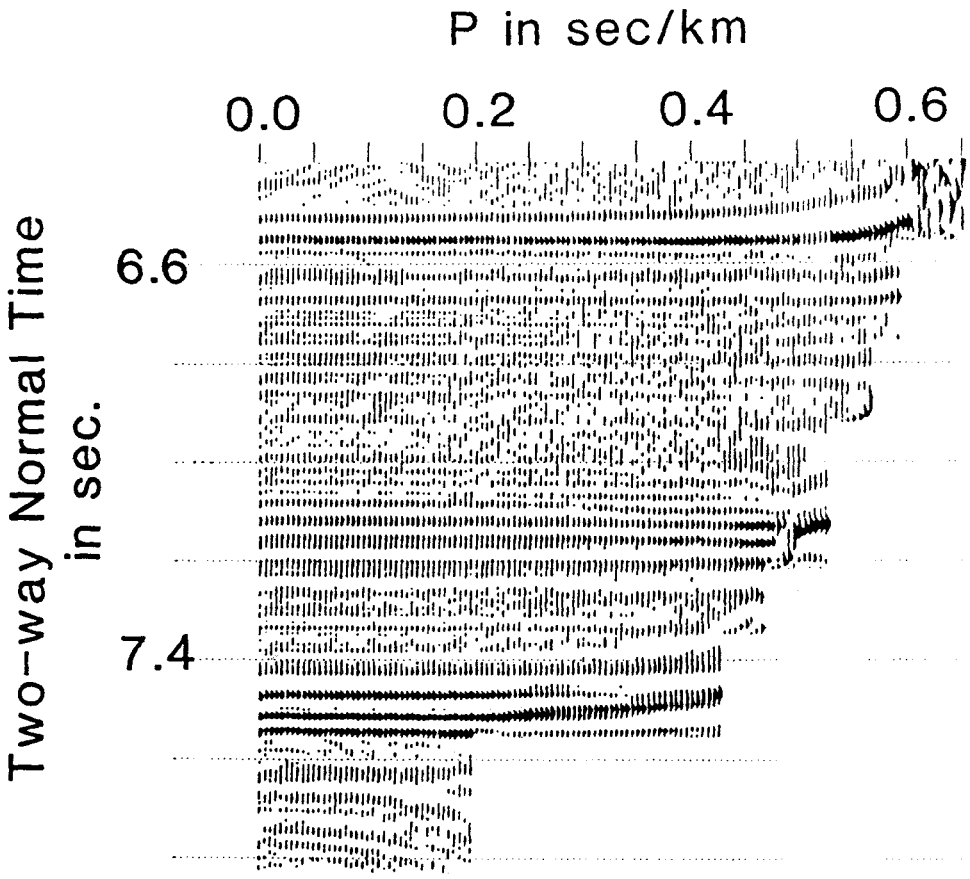


Figure 7. The completely moved out gather.

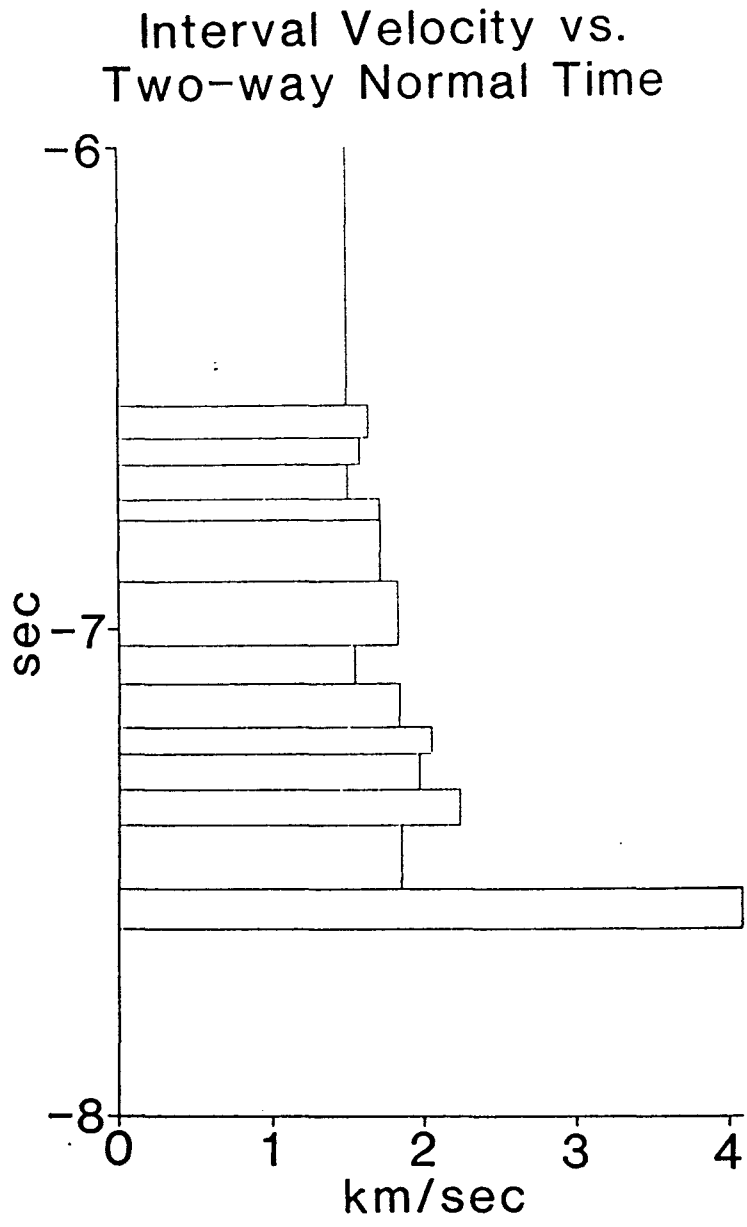


Figure 8. Interval velocity vs. two way time for the final model.

Appendix

I. OPERATION OF THE τ-p INTERACTIVE VELOCITY ANALYSIS PROGRAM

The program is invoked by the command:

ppick datafilename modelfilename

If a model file name is given then the file must look like this:

```
1                                          ! gather number
1       8                                  ! analysis number, number of layers
1500    4.875   0.0     1.0                ! layer velocity, thickness, dip, and density
1582     .543   0.0     1.0
.        .       .       .
.        .       .       .
.        .       .       .
5000    5000    0.0     1.0
```

The parameters of dip and density are reserved for future versions of the program. The last layer of the model must be the underlying half-space.

When invoked the program will display a sampling of the traces which span the range of τ and p. If a model was provided the model curves will also be displayed. The analysis works on one layer at a time, the selected layer, which is indicated in the upper left corner of the screen. To add a layer select L-ADD. This will add a layer above the selected layer. Once a layer is selected, the τ_0 time is picked by placing the cursor at the desired time and pressing mouse button 1. Velocity is adjusted by pressing mouse button 3 on the lower portion of the screen for a low velocity or higher on the screen for a higher velocity. The velocity range is adjusted by pressing mouse button 1 or 2 at the bottom of the screen. Button 1 decreases the upper velocity bound by a factor of 1.5 and button 2 increases it by a factor of 1.5.

When the model curve matches the seismic data the gather can be moved out to either time or depth by selecting either NMO-T or NMO-D. If a mistake is made the raw gather can be recovered at any time by selecting RAW. The current model can be saved at any time by selecting SAVE and the model will be written to a file called "test.out'. Plotting parameters can be adjusted by selecting PLOT. "Trace mod" is the no. of traces skipped between traces. For example trace mod = 5 means that only every fifth trace of the gather will be drawn. This option is useful for quickly getting a rough idea of the data. Mouse button 1 decrements the trace mod and mouse button 2 increments it. Mouse button 3 controls wether the display will be;

a) wigl wiggle trace

b) vaw variable area fill wiggle trace

c) va variable area fill

+va or +vaw fills to the right and -va or -vaw fills to the left.

To window the data select WINDOW and press mouse button 1 where the upper left corner of the new display is to be and press mouse button 2 where the lower right corner is to be. Button 3 performs the windowing. Pressing button 3 before pressing 1 or 2 will re-draw the entire gather.

II. MENU DESCRIPTIONS FOR P-PICK

A. Above the data window

Menu Option	Mouse Button	Description
LAYER	1:	Decrement number of selected layer
	2:	Increment number of selected layer
	3:	Re-draw the current model
WINDOW	1:	Pick upper left corner of sub-window
	2:	Pick lower right corner of sub-window
	3:	re-draw the entire window
RAW	any:	re-draw the raw τ-p data and current model
NMO-T	any:	Apply NMO in time
NMO-D	any:	Apply NMO in depth
SAVE	any:	Write current model as file "test.out"
L-ADD	any:	Make a new layer before existing selected layer
L-DEL	any:	Delete selected layer
1D-2D		reserved for future implementation
PLOT	1:	Decrement traces skipped between traces drawn
	2:	Increment traces skipped between traces drawn
	3:	Choose wiggle trace and or area fill options
QUIT	any:	Exit from program

B. Within the data window

Mouse Button	Description
1:	Select p=0 time for selected layer
2:	Type out current parameters for selected layer
3:	Pick new velocity for this layer by clicking high or low on the screen. The model will be re-drawn.

C. Below the data window

Mouse Button	Description
1:	Decrease maximum velocity by a factor of 1.5
2:	Increase maximum velocity by a factor of 1.5

Anywhere: press **cntrl /** to stop drawing traces and return control to the program.

Preliminary Interpretation of the 3D Structure of an Accretionary Wedge off Costa Rica

Donald F. Dean

ABSTRACT

The University of Texas Institute for Geophysics conducted a 3D marine multichannel seismic survey offshore Costa Rica in April 1987. The data were acquired by the R/V *Fred H. Moore* using a turned source array, a 96 channel 3.5 km receiving array equipped with compasses, and shore based navigation. Navigation reduction and data processing started in the summer of 1987. The objective of the study was to define the tectonic processes responsible for continental margin accretion along the Middle America Trench subduction zone. Landmark picture files will be used to show structural features and the internal geometry of the accretionary wedge not previously observed in conventional seismic data.

University of Texas at Austin, Institute for Geophysics, 8701 Mopac Boulevard, Austin, Texas 78759-8345

INTRODUCTION

Accretionary prisms at convergent margins are zones of complex geologic structure which are difficult to image. A number of schematic models have been proposed to explain the growth of accretionary prisms, but the internal geometry is poorly imaged with conventional multichannel techniques. The University of Texas Institute for Geophysics conducted a 3D marine multichannel seismic survey offshore Costa Rica, in April 1987, the location is shown in Figure 1. The 3D program was designed to image the complex tectonic structures and to define the tectonic processes responsible for continental margin accretion along the Middle America Trench subduction zone. Two types of data were collected and processed.

The first data set consists of a series of 9 dip (swath) lines that extend from the shelf edge to the trench axis. These lines were 60 km long and spaced symmetrically by a distance of 250, 750, 2000, and 4500 m about a reference dip line. These data form a swath of seismic lines that are designed to image the entire cross section of the margin including the accretionary prism complex. These data were shot at 33 m interval resulting in 48 fold CDP data.

The second data set is a 3D grid survey which is composed of eighty-eight 23 km lines spaced every 100 m with a shot interval of 33 m. This survey is located over the most deformed zone of the accretionary prism complex. The 3D grid was divided into two data sets. The first data set was composed of just the six nearest traces. The second data set contained all 96 traces. This division was made to simplify the initial 3D grid data processing and provide a data set suitable for the initial interpretations.

The data were acquired with the R/V *Fred H. Moore* and the field parameters are shown in Table 1. Navigation reduction and data processing were carried out at the Institute for Geophysics and started in the summer of 1987. All seismic processing was done on the University of Texas

Center for High Performance Computing Cray X-MP/24 using CGG's Geovecteur seismic processing software. The processed data were then displayed on the Landmark workstation for interpretation. The processing parameters are shown in Table 2. The following Landmark picture files show structural features and the internal geometry of the accretionary wedge not previously observed in conventional seismic data.

Picture File 1

This is a portion of one of the swath lines, CR80, that extends from the shelf edge to just beyond the trench axis. The oceanic crust on the left is being subducted at a rate of about 9 cm/yr. The normal faulting in the oceanic crust is due to extension caused by the plate bending during subduction. The prism or accretionary wedge is the area bounded by the horizontal reflections at around 6 seconds and the rough surface above at about 5 seconds near the trench to around 4.2 seconds up the slope. Imaging of the internal structure of the prism is very difficult due to the very rough surface at the top of the prism.

Picture File 2

This is a portion of swath line CR80. The area outlined in white is the toe of slope and trench axis where the initial deformation takes place.

Picture File 3

Blow-up of outline box in Picture File 2. There is a minor amount of offscraping of oceanic sediments at the toe of the slope. Note the change in polarity with respect to the sea floor reflection just landward of the trench axis where it continues under the prism complex. This reflection is from the decollement zone. The decrease in acoustic impedance is likely a result of the subduction of low velocity oceanic sediments beneath tectonized slope or accreted sediments, and by fluids escaping from the compaction of the oceanic sediments on the downgoing slab as the plate is subducted.

Picture File 4

This is a portion of swath line CR80. The white outline shows the location where the 3D survey was done. This area of the accretionary wedge is the most deformed and poorly imaged due to the rough surface at the top of the prism.

Picture File 5

This is a portion of 3D grid line 81. The data have been 3D binned and stacked but not migrated. Note the numerous diffractions that come from the top of the prism that mask the internal reflectors in the prism.

Picture File 6

This is the same line as in Picture File 5 after a 2D x 2D time migration. The top of the prism is more clearly defined and internal structures within the prism are imaged for the first time.

Picture File 7

This is a portion of 3D grid line 31. The data have been 3D binned and stacked but not migrated. Again there are numerous diffractions that come from the top of the prism that mask the internal reflectors in the prism. There is also a feature on the seafloor around CDP 950 that is interpreted as a fluid escape structure or mud volcano.

Picture File 8

This is the same line as in Picture File 7 after a 2D x 2D time migration. The top of the prism is more clearly defined and structures within the wedge are now imaged. These dipping structures may be thrust fault boundaries or pathways for fluids deep within the wedge. The small arrow indicates what may be a fluid migration pathway feeding the mud volcano.

Picture File 9 and 10

This is a comparison of the 6-fold 3D grid (line 172), Picture File 9 and the full-fold 3D grid (line 31), Picture File 10 in the area of the mud volcano. The data are unmigrated and show many diffractions. The full-fold data contain greater signal level and more reflections within the slope sediment cover.

Picture File 11 and 12

This is the data from Picture Files 9 and 10 after a 2D x 2D time migration. Picture File 11 is the 6-fold data and Picture File 12 is the full-fold data. The mud volcano is very well defined along with the fluid conduit that may be feeding it (arrow).

Picture File 13

This is a horizon map of the seafloor of the Costa Rica 3D grid. This was obtained by digitizing the seafloor on all 88 lines. Note the location of the mud volcano shown by the arrow.

TABLE 1. COSTA RICA 3D FIELD PARAMETERS

- 96-trace streamer
- 33.33 meter groups
- 10 compasses
- Radar tracking of tail buoy

- 429, 240, 150, 108, 78, 65, cubic inch air gun array
- 1065 cubic inches, 2000 PSI, 23 bar-m @ 8-128 Hz
- 33.33 meter shot spacing
- Maxiran shore-based navigation system
- Global positioning system WGS-84 calibration

- 9 swath lines, 60 km long
- 250 m, 750 m, 2000 m, 4500 m, from center line

- 12 'strike' lines, 14 km long, spacing 2000 m

- 88 'grid' lines, 23 km long, spacing 100 m
- 16 days to acquire

TABLE 2. COSTA RICA 3D PROCESSING

Low fold 3D, 6 near channels, offsets 170 to 337 m

- Deconvolution every trace
- 3D binning 100 m x 33.33 m, 6 fold
- NMO and stack--60,000 output traces
- 2D x 2D finite difference time migration

Full fold 3D, 96 channels, offsets 170 to 3337 m

- Deconvolution every trace
- 3D binning 50 m x 33.33 m, 80 fold
- Velocity analyses every 1 km x 2 km
- NMO
- Partial stacks for 12 offset ranges
- Full stack--120,000 output traces
- 2D x 2D finite difference time migration
- 3D depth migration

Special processing

- Evaluation of 2D swath vs. 3D grid
- Evaluation of 6-fold vs.80 fold
- partial dip moveout improvements

Special interpretation

- Geological velocity estimates from pre-stack migration
- Investigation of phase reversals
- Offset dependant amplitude studies
- Structural interpretation of 200 km^2 on a Landmark Workstation

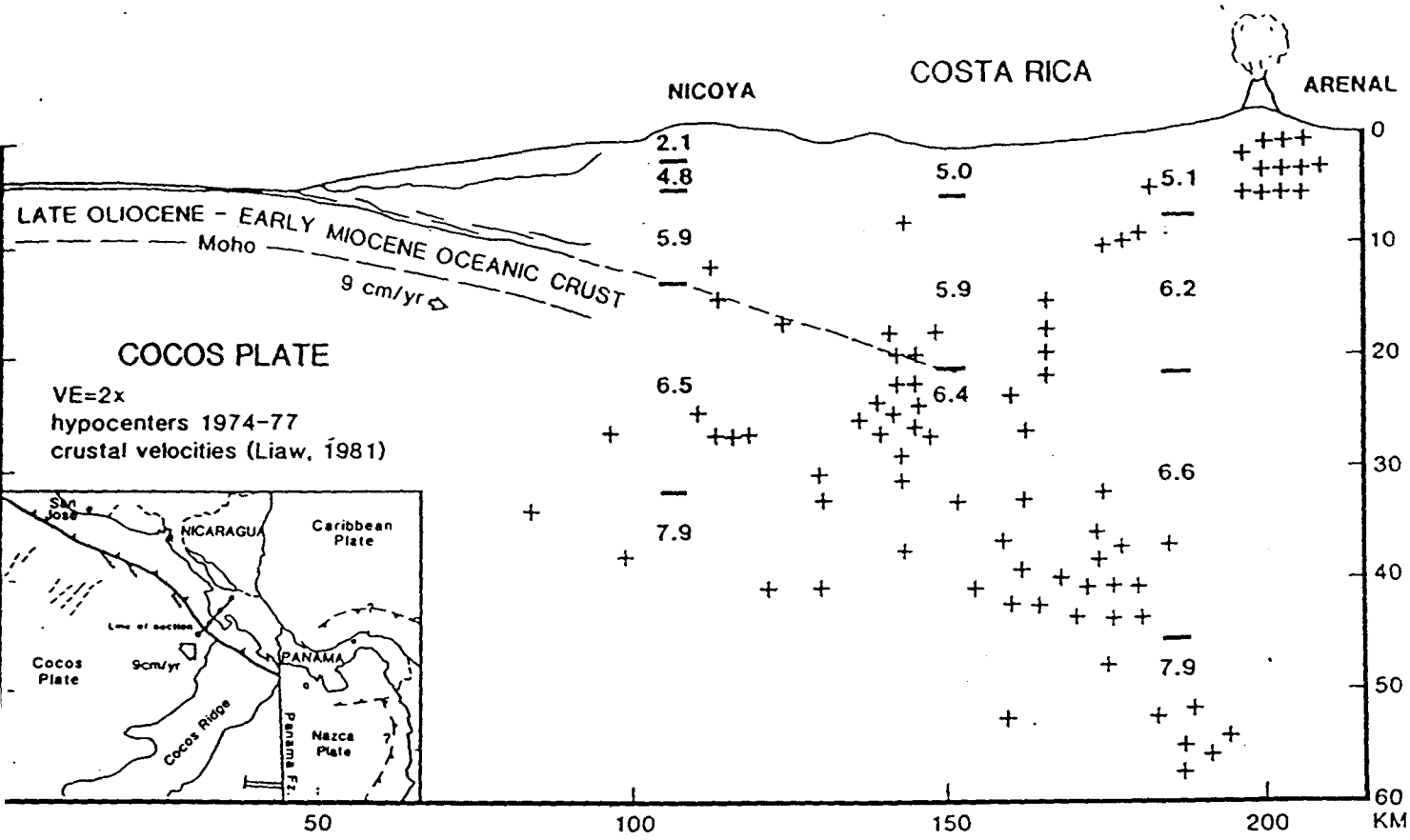
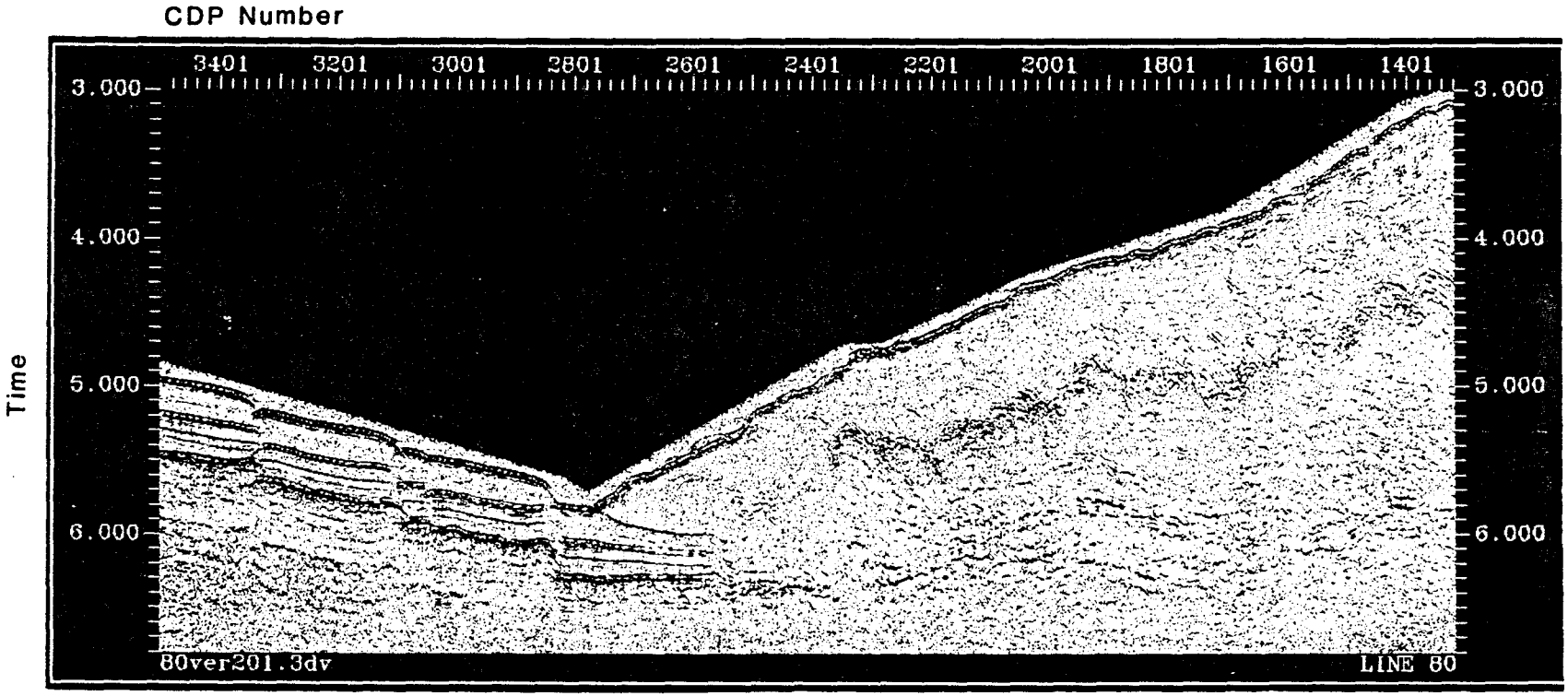
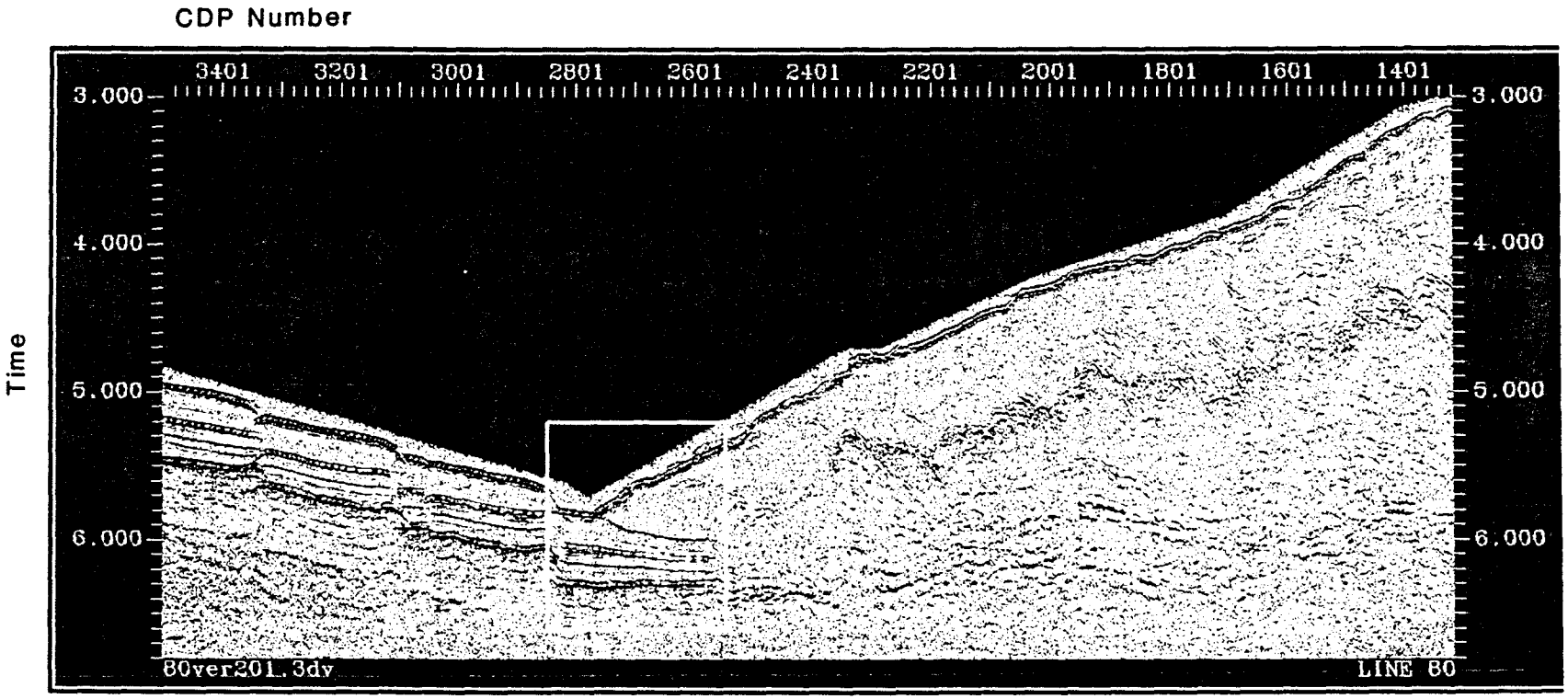


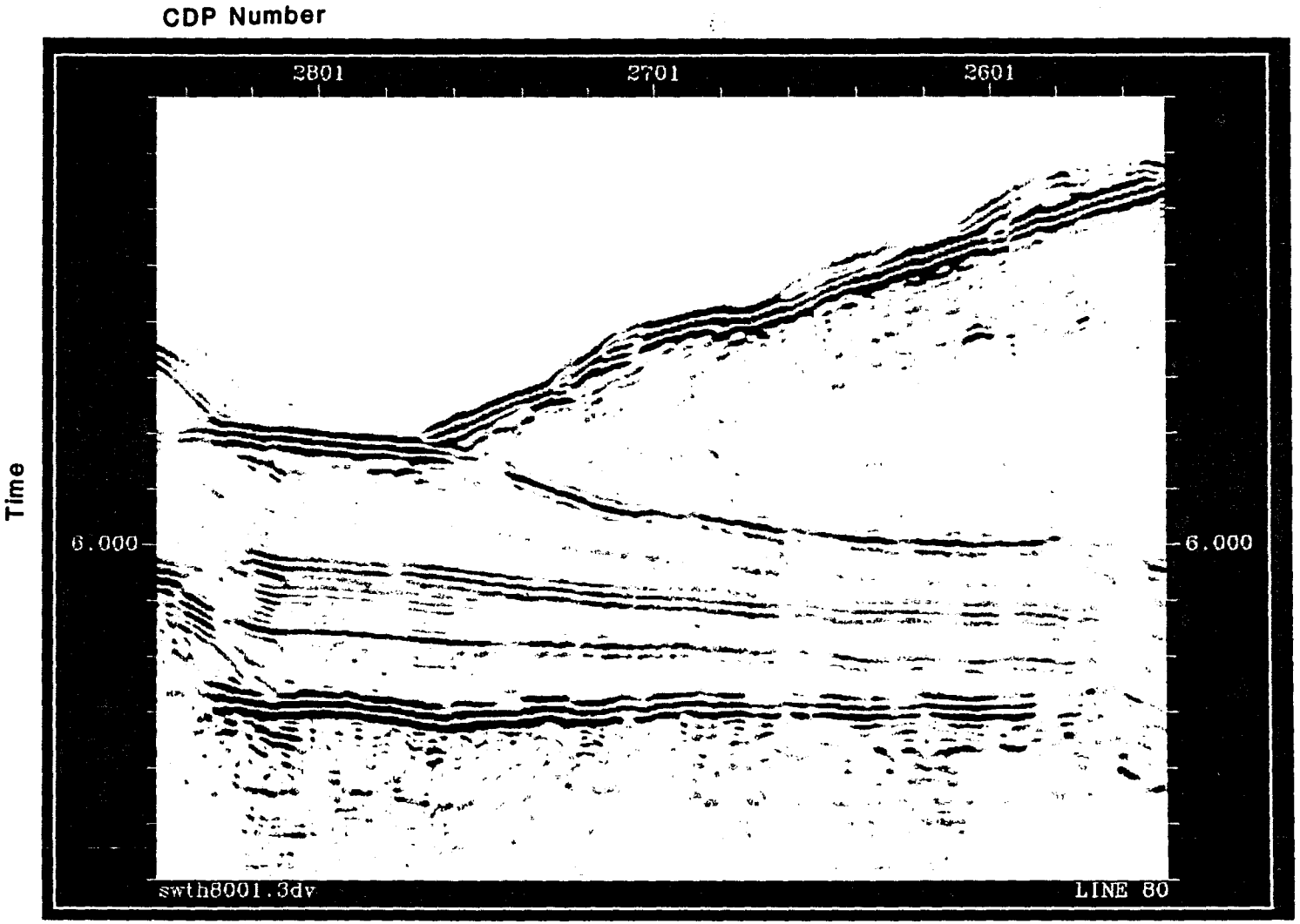
Figure 1. General location and cross section across the continental margin and arc of Costa Rica (modified from Liaw, 1982 and Crowe and Buffler, 1983). The trench axis is about 50 km from the shoreline. Line of section is shown in inset.



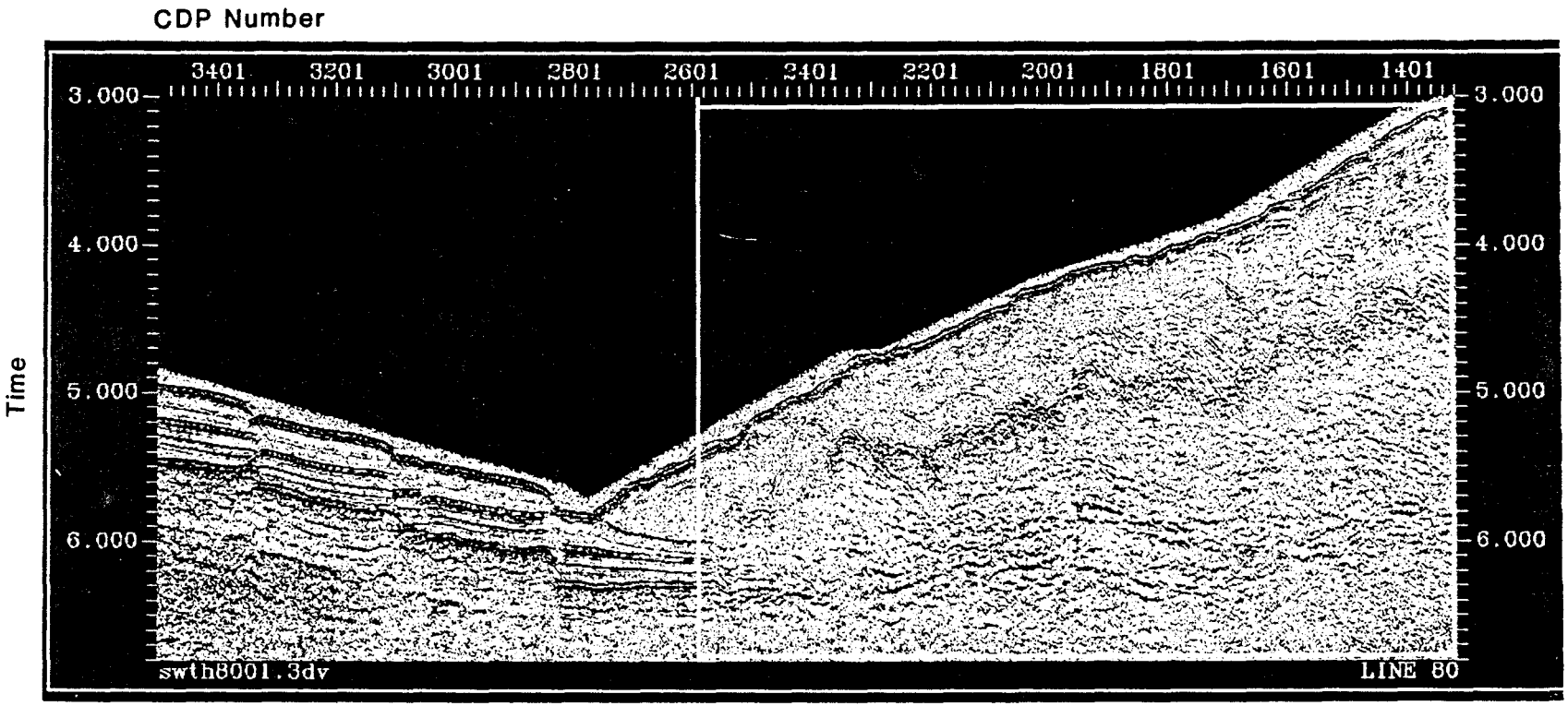
Picture File 1



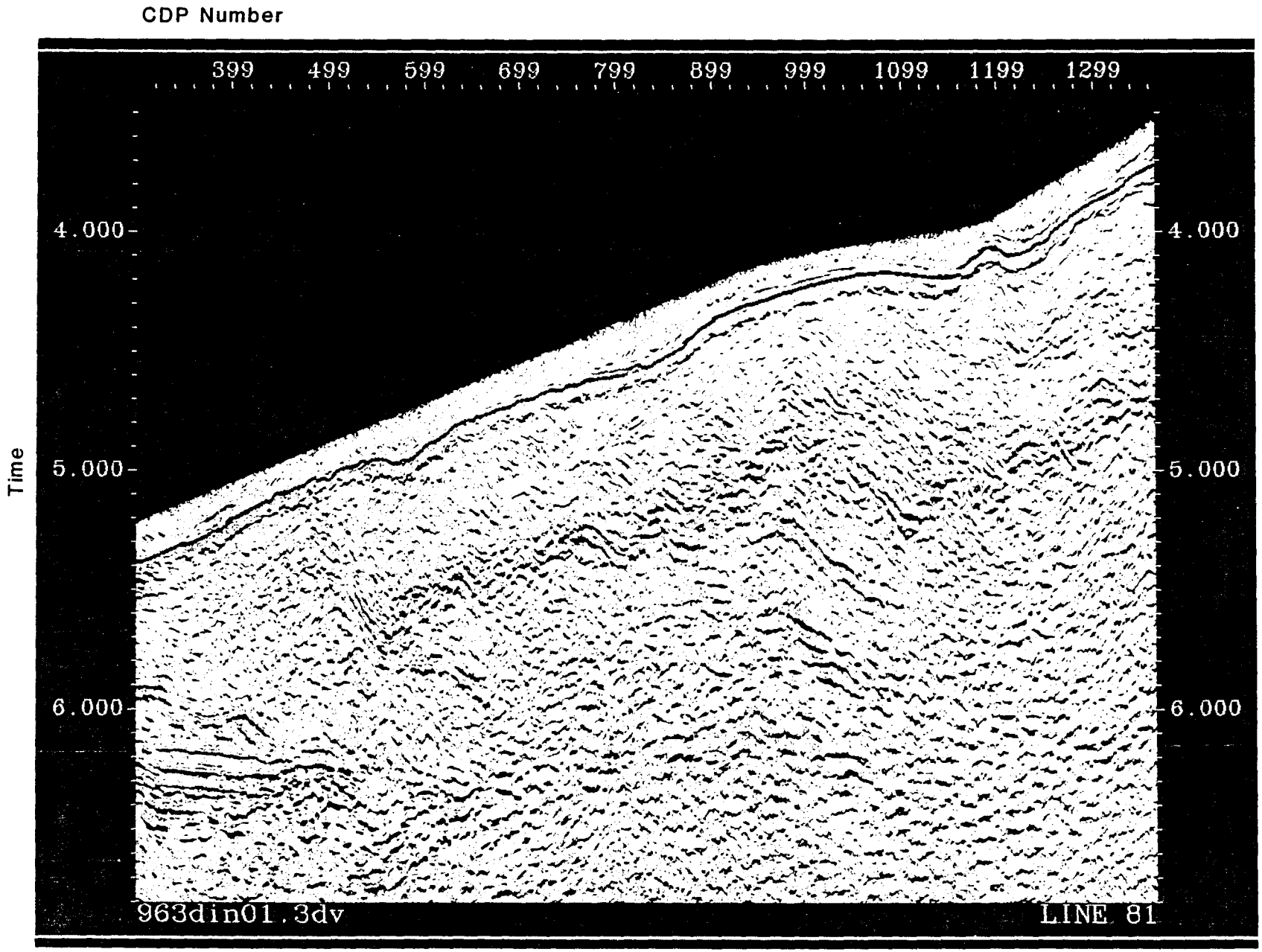
Picture File 2



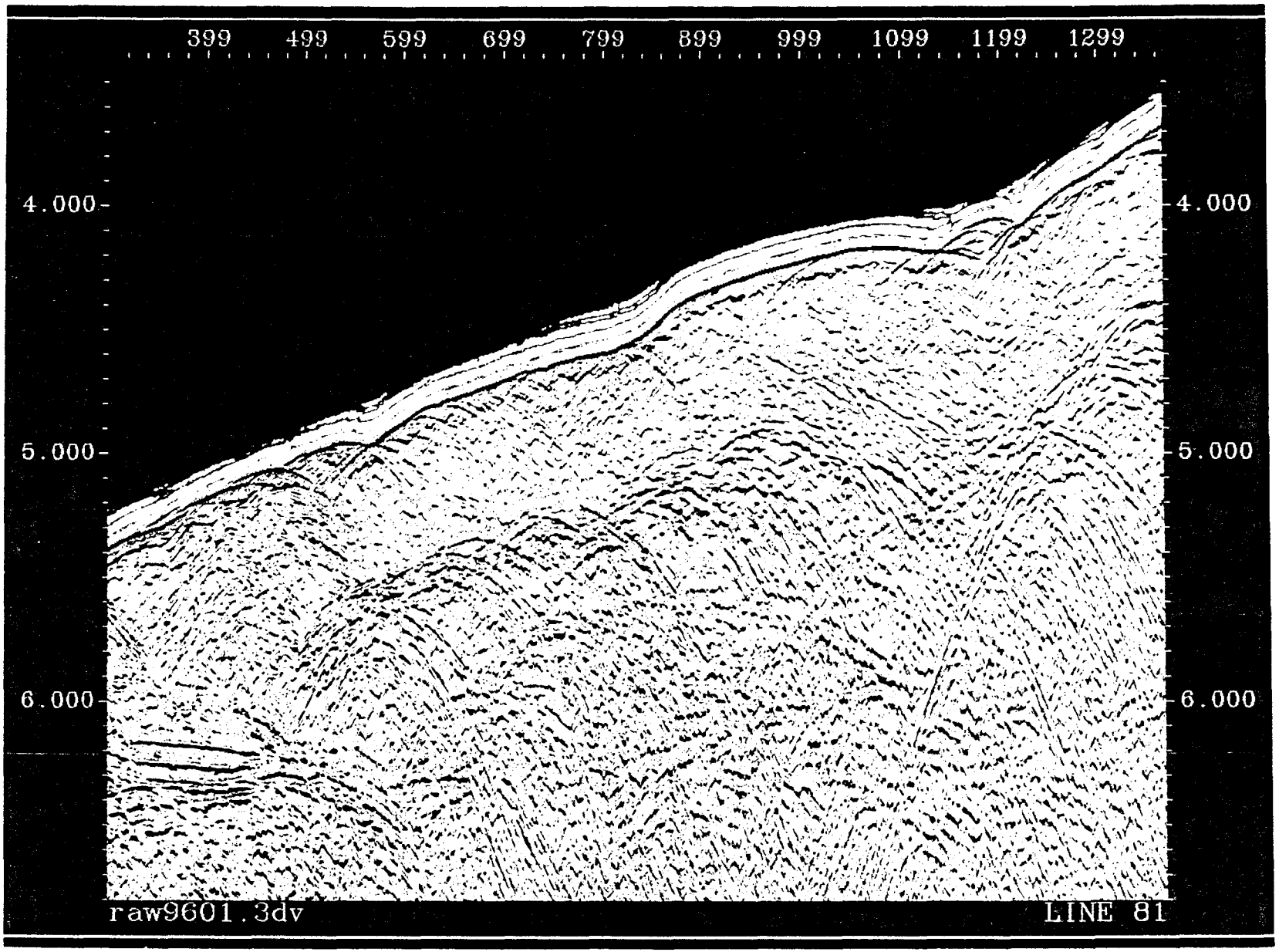
Picture File 3



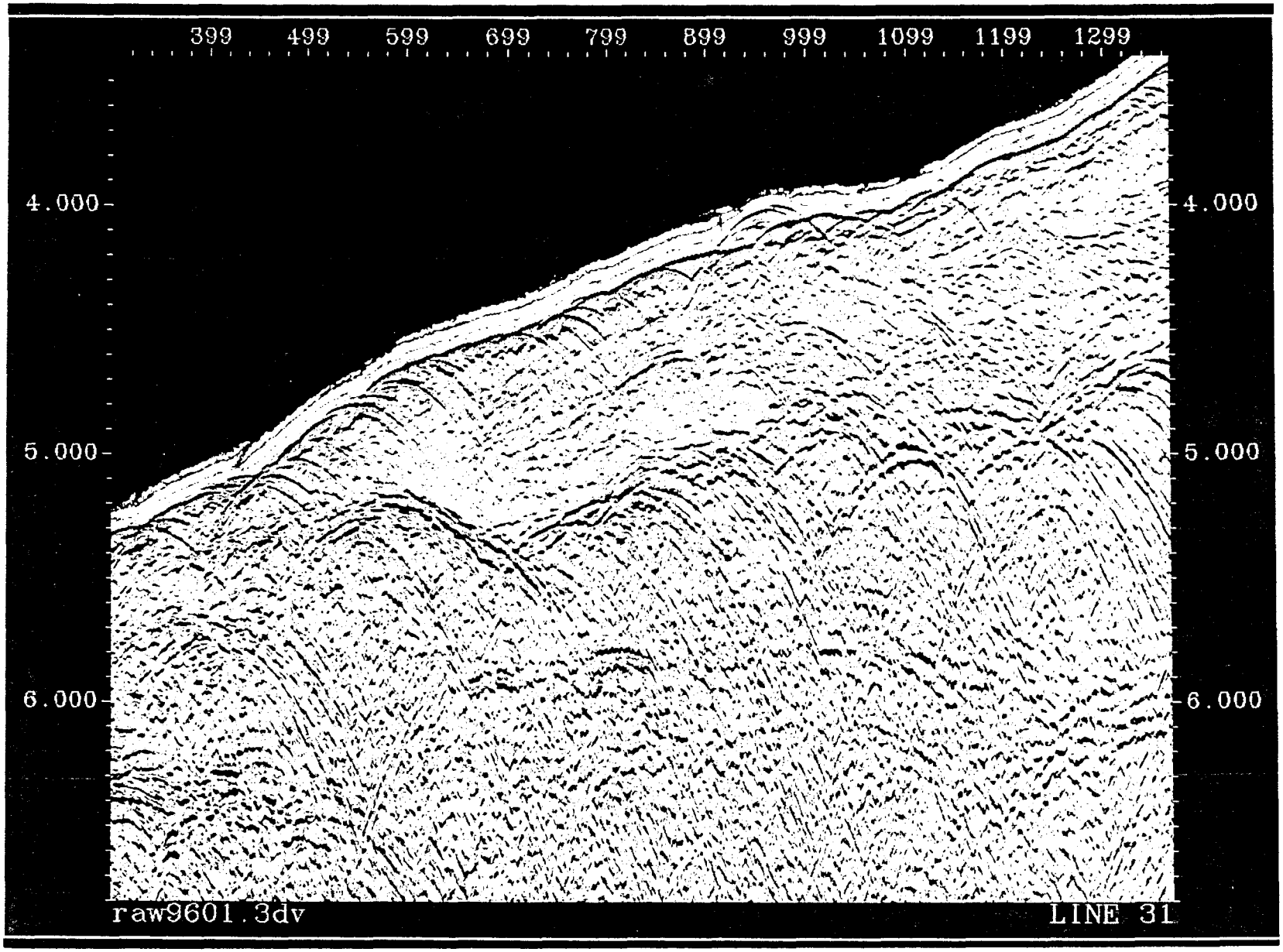
Picture File 4



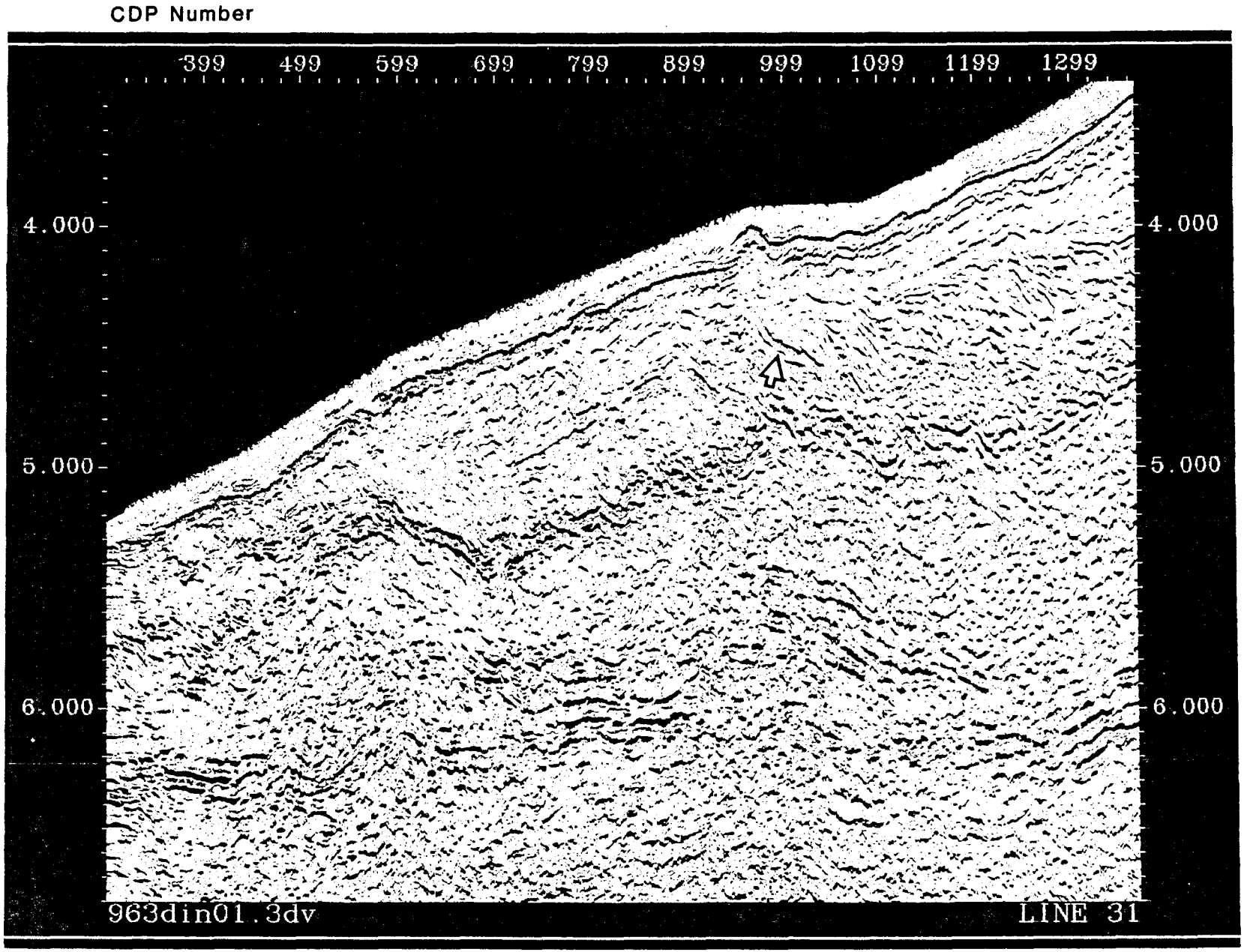
Picture File 5



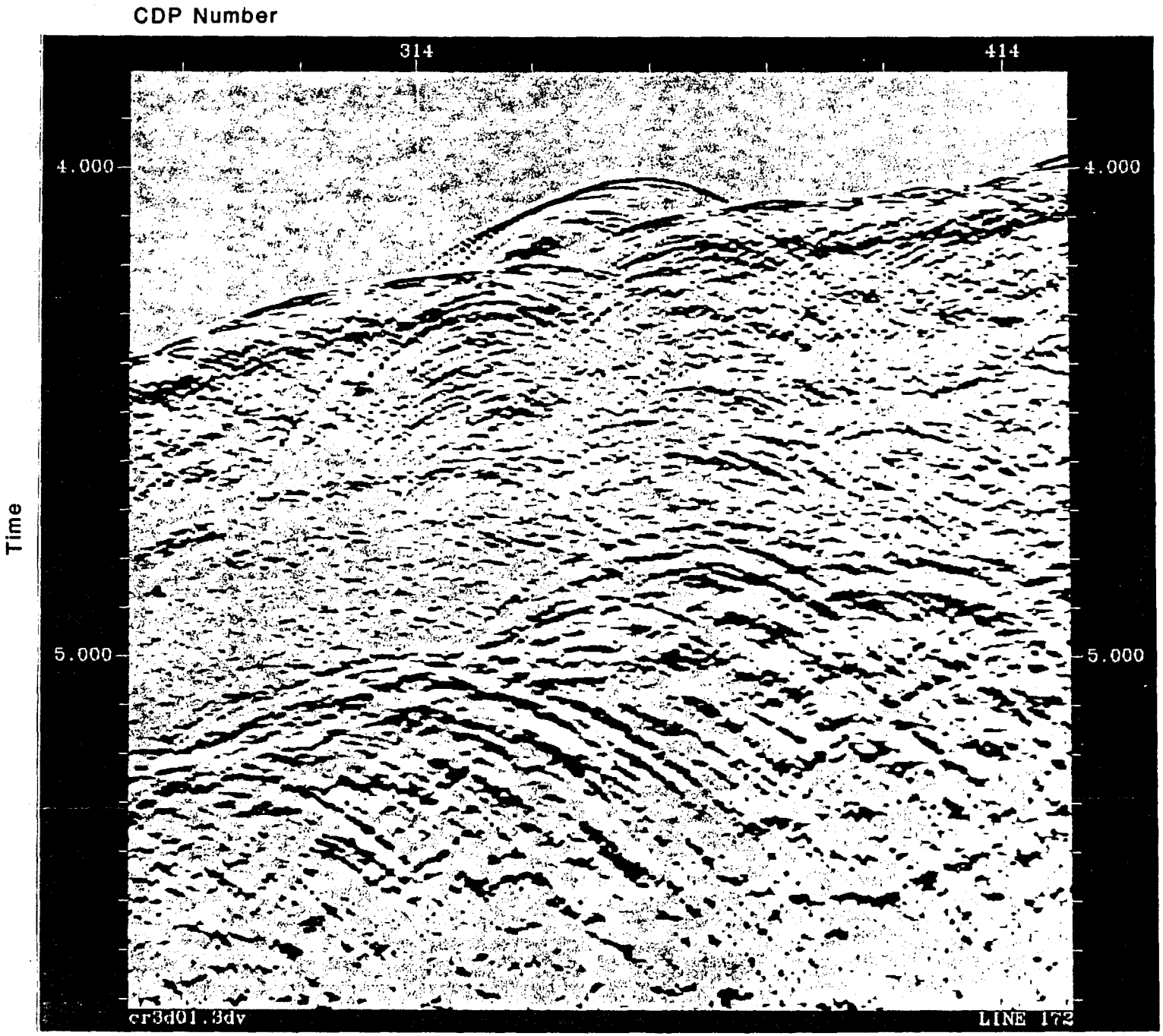
Picture File 6



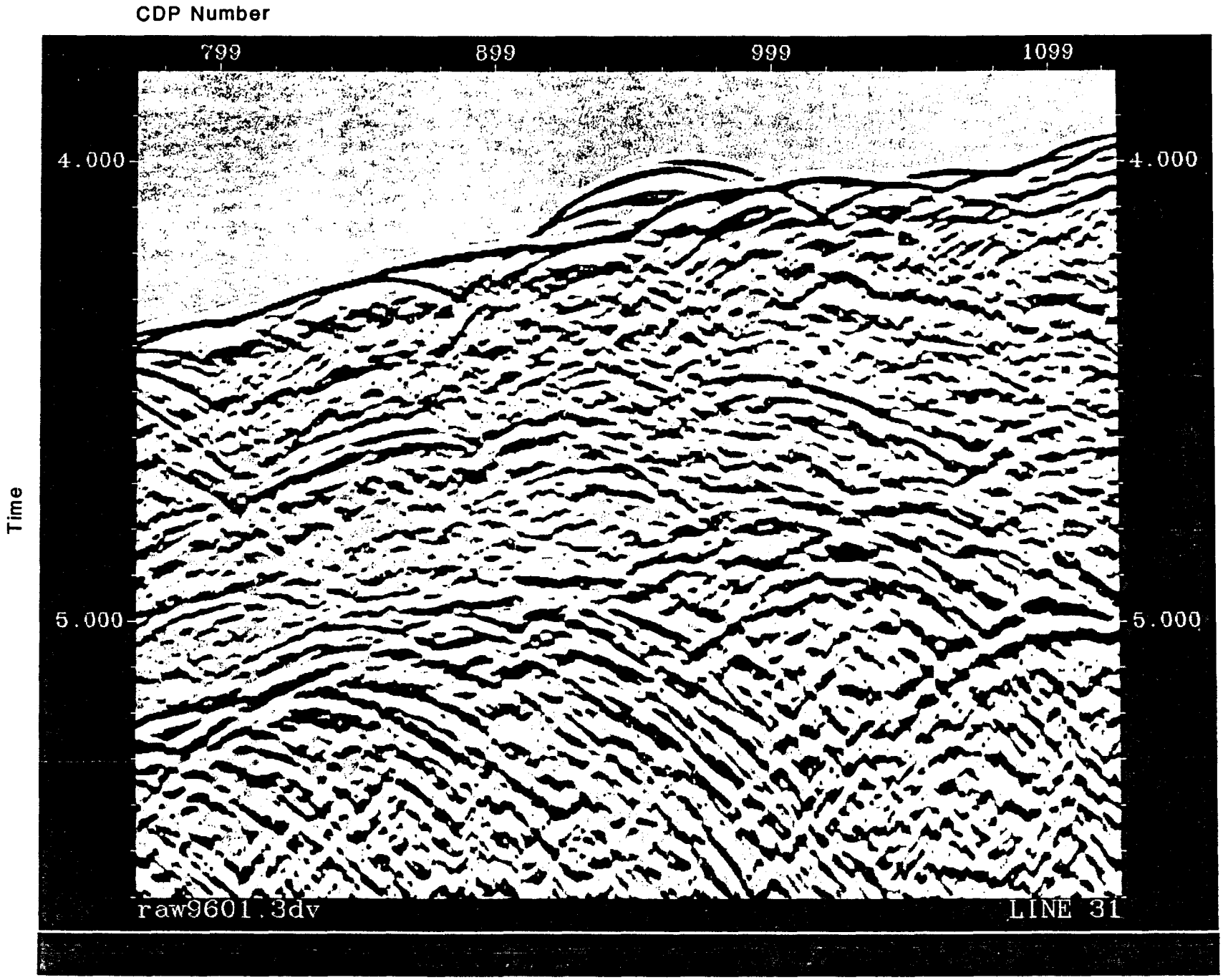
Picture File 7



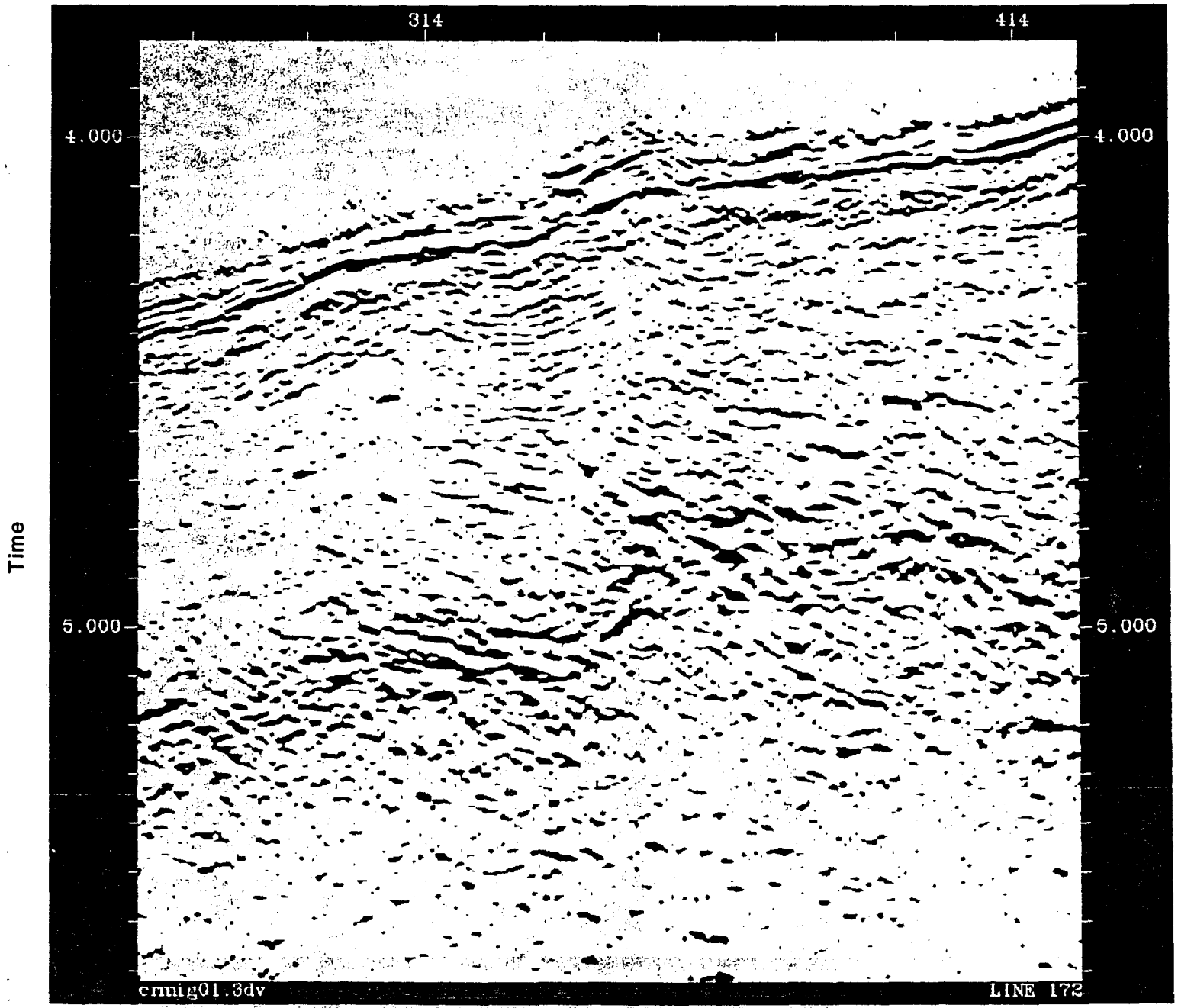
Picture File 8



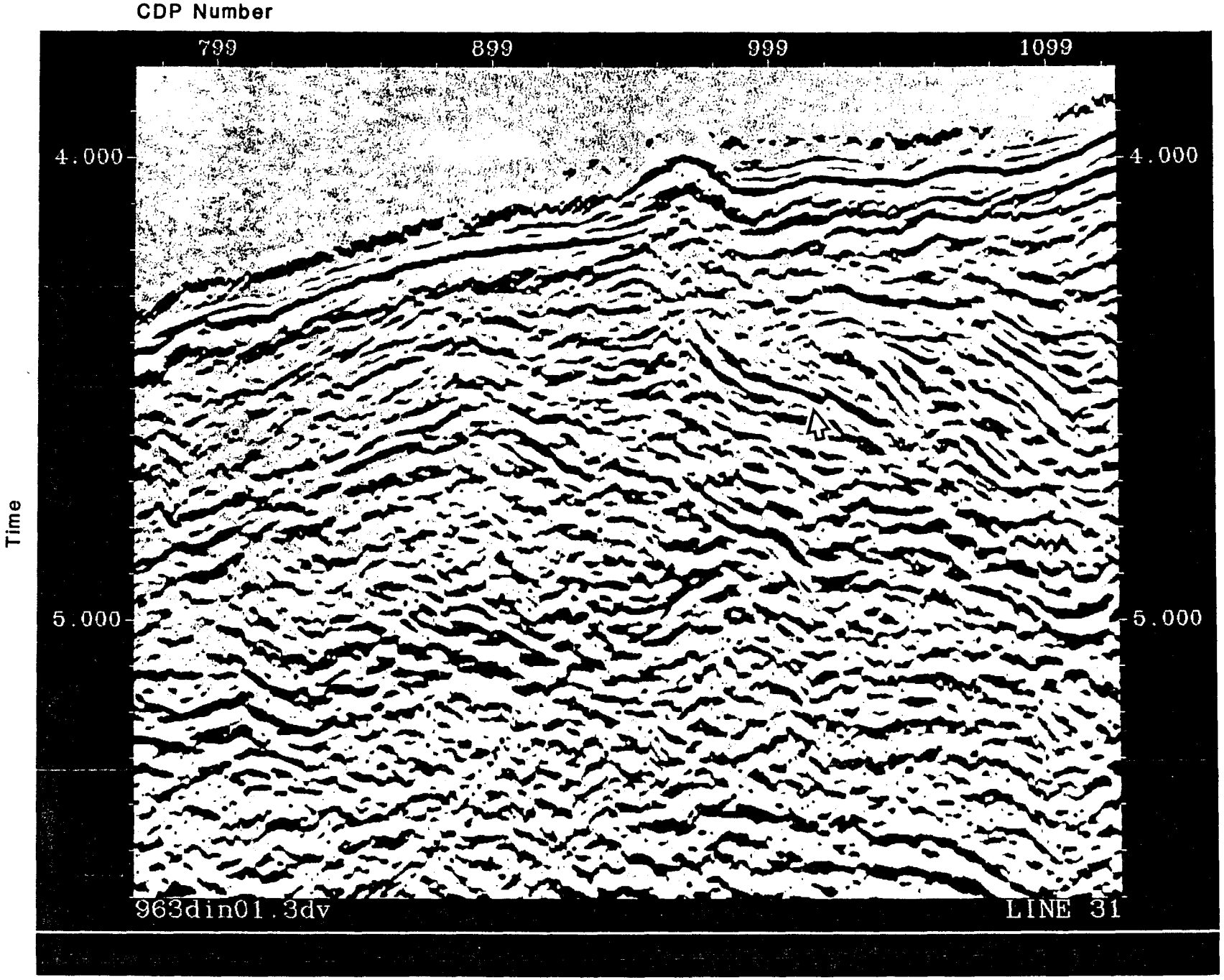
Picture File 9



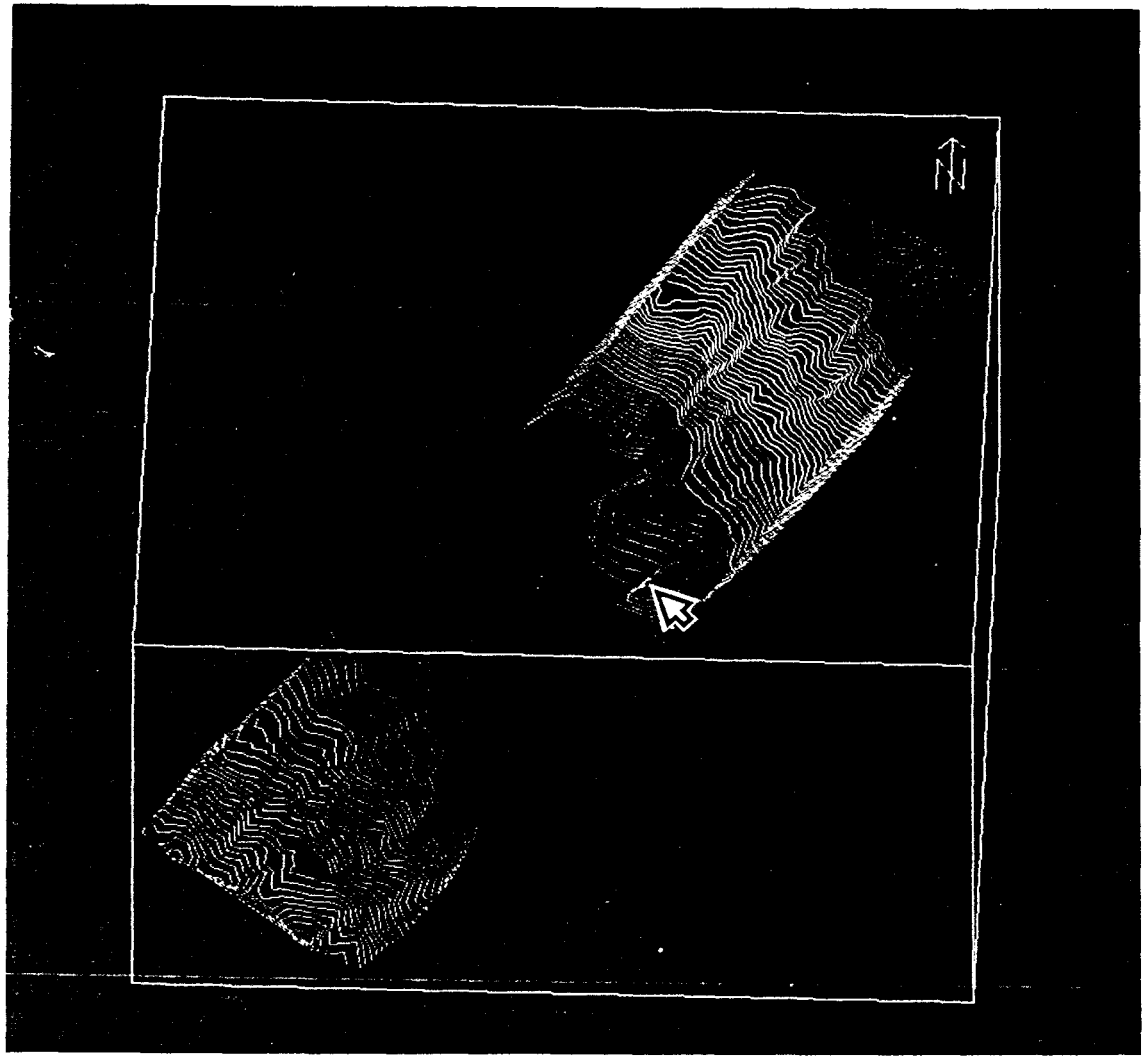
Picture File 10



Picture File 11



Picture File 12



Picture File 13

SEG-Y File Access Package

Mark Wiederspahn

ABSTRACT

The seismic exploration community has used a standard tape format called SEG-Y for seismic information interchange since the standard was published in 1975. It is well accepted in part because it is not complex and is well adapted to magnetic tape characteristics. There is no equivalent disk storage format standard; this becomes a liability in a mixed vendor networking environment. This SEG-Y disk access package gives the user access to SEG-Y format data on either tape or disk, and when properly implemented on each host, permits such files to be transparently accessed over the network. It has been implemented under Cray/COS, IBM RT/AIX, Masscomp/RTU and Sun/SunOS.

University of Texas at Austin, Institute for Geophysics, 8701 Mopac Boulevard, Austin, Texas 78759-8345

SEG-Y ON DISK WHY? WHAT? HOW?

Why:

The Institute for Geophysics currently has a variety of computers available. Each is attached to an ethernet which appears to span the entire campus. Figure 1 shows our local computing environment consisting entirely of Unix based machines, including a Landmark workstation. None of these computers has the speed for serious seismic processing, however; for that we turn to the UT System Center for High Performance Computing Cray(s) (Figure 2). Although this center is also on the same network, it is about 2.5 miles away as the crow flies. Quite often, we want to locally manipulate final results on our local computers. Although the bandwidth of boxes of tapes in the back seat of a car is quite high, (about 3.3 Mb/sec if you carry 4 boxes a trip!) nobody wants to go out in the rain. Transferring disk files by network is an obvious answer. Segy is also an obvious format. There is no standard format for disk or network form for segy, however. This paper describes some of the issues, and how we choose to solve them.

What:

The standard segy structure on magnetic tape is quite simple. There is a 3200 byte reel header consisting of 40 80 byte EBCDIC format records, followed by a 400 byte binary format reel header, then zero or more data records, and then one or more end of file marks. Data traces are themselves comprised of a 240 byte binary trace header prefixed to a variably sized time series. Each of these entities is separated from any other by an inter-record gap consisting of erased tape. This format is well defined for the tape medium, but could be mapped to disk in several ways.

The first issue is how to mimic the tape record structure. Many systems include a count byte before each record (VMS) but some use a flag word instead (COS). One scheme might be optimal

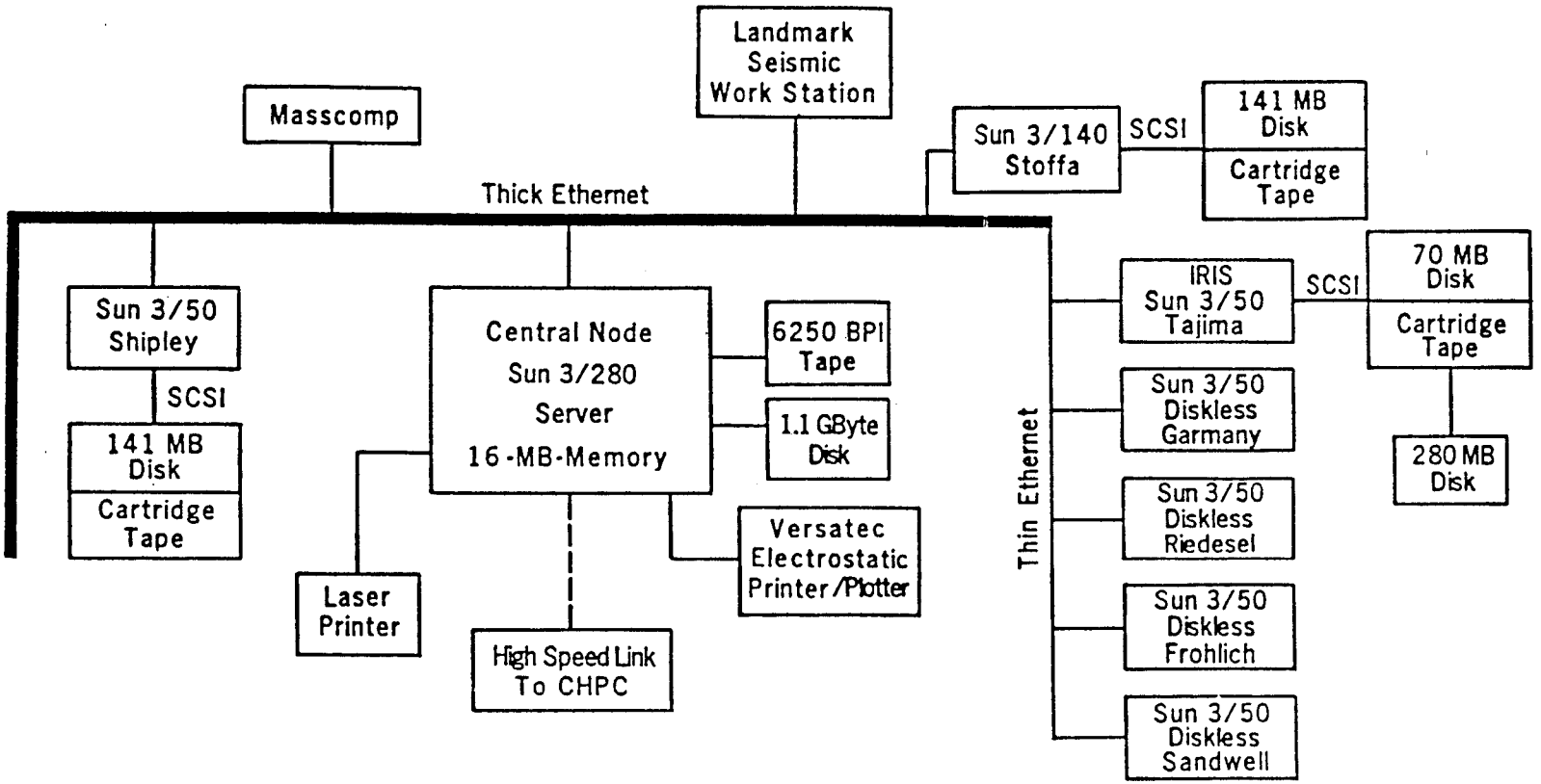

Landmark
Seismic
Work Station
Masscomp
Sun 3/140
Stoffa
SCSI
141 MB
Disk
Cartridge
Tape
Thick Ethernet
Sun 3/50
Shipley
SCSI
141 MB
Disk
Cartridge
Tape
Central Node
Sun 3/280
Server
16-MB-Memory
6250 BPI
Tape
1.1 GByte
Disk
Laser
Printer
High Speed Link
To CHPC
Versatec
Electrostatic
Printer/Plotter
Thin Ethernet
IRIS
Sun 3/50
Tajima
SCSI
70 MB
Disk
Cartridge
Tape
280 MB
Disk
Sun 3/50
Diskless
Garmany
Sun 3/50
Diskless
Riedesel
Sun 3/50
Diskless
Frohlich
Sun 3/50
Diskless
Sandwell

fig. 1

UT System Center for High Performance Computing
CRAY X-MP/24, X-MP EA/14SE SUPERCOMPUTERS AND SUPPORT COMPUTERS CONFIGURATION

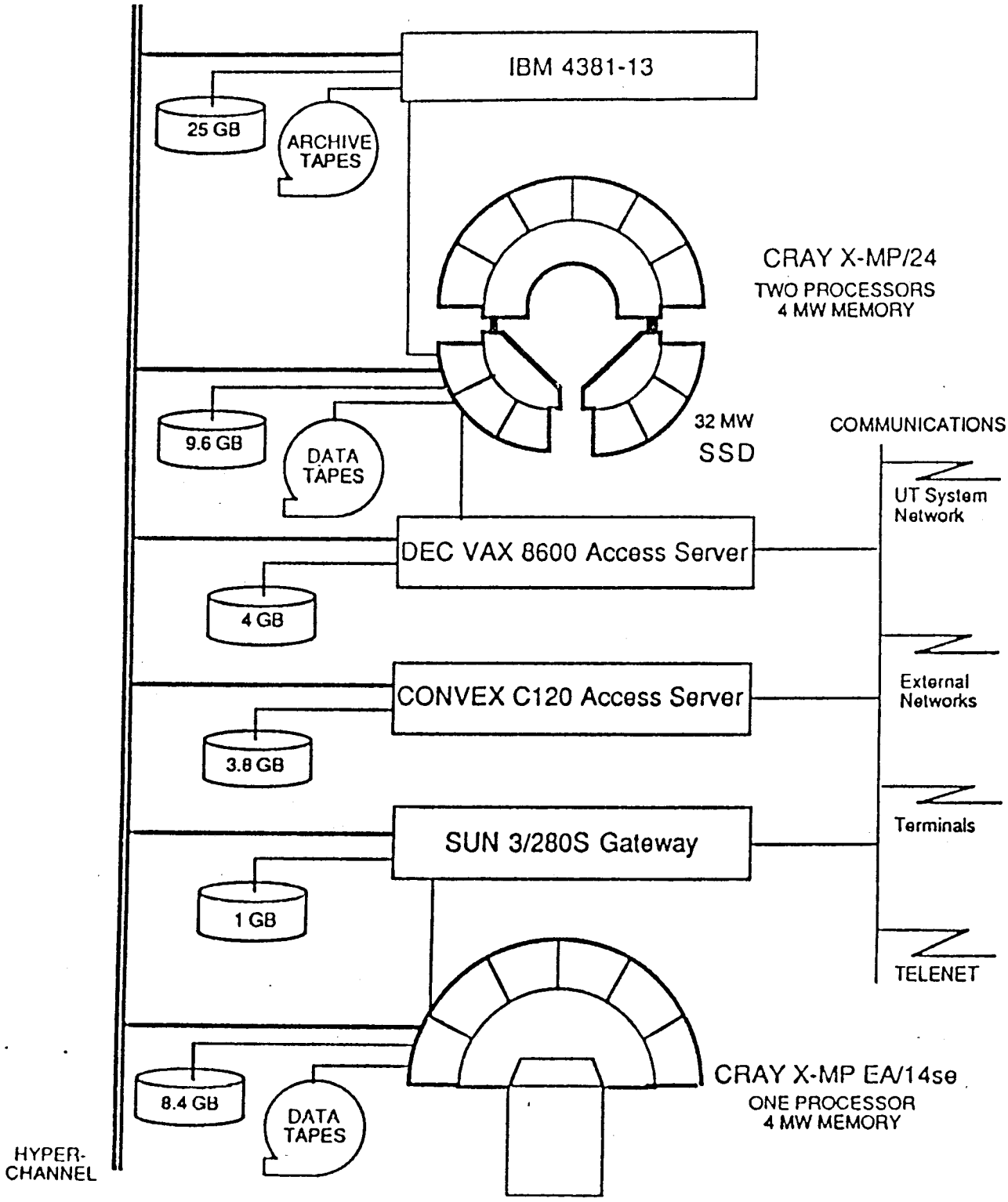


fig. 2

for a particular machine, but a mess on some other. Another alternative is to squash the file, so that the inter-record gaps just disappear. This is what happens when you read in a tape file using unix "dd", for example. This is simple, and is portable, but may lead to lower performance on some systems due to traces not starting on physical disk block boundaries. A modification of this scheme would start each record on a disk block boundary, with zero padding at the end of each record out to a disk block boundary, as needed. This is how Cogniseis DISCO stores its internal format files. This is a high performance solution which is close to hardware, but a significant problem is a common disk block size for all implementations. Some optical disks have unusually large minimum requirements.

The next next storage issue is the internal order of binary data. When 8 bit bytes are stored on tape, thery are atomic; there is only one way of sequentially accessing them. When these are aggregated into larger structures like integer*2 or integer*4 data, there are several ways to order them. The segy standard orders them in "big-endian" fashion, meaning the bit values of the first data of a value are the highest in the construct. This is usually called "ibm" order. The alternative order is usually called "dec" order, although it is also called "little-endian" and "intel 8086" order. Whichever format we choose, we will displease half of our users, so we choose the "ibm" standard segy order ALWAYS.

The internal format of the data traces themselves is relatively standard. We should allow all of these standard forms (i*2, i*4, ibm floating point) but may also support unusual ones like 1 byte integers or double precision floating point. In a network environment with low bandwidth or when we are archiving data for infrequent access we might consider compression schemes similar to those used for earthquake seismology, which reduce the bits/sample during portions of the trace with low slew rate. These can obtain 3.5:1 reduction in storage requirements, but with an increase in retrieval time.

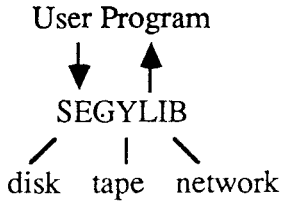
The issue of input/output efficiency has already been raised. We want to read or write as few times as possible, and don't want to have to pack or unpack records from the disk file. In some cases (most versions of unix, for example) this is done by the system in any case. Most high performance systems let you read or write directly from memory to the device, but usually only on block boundaries. We must trade off generality for performance here.

The special cases of reading or writing directly over the network should also be considered. The issue of bandwidth has been mentioned. Many network protocols may impose flow control which can cause a partial return of data. This means that data is available, it is just not available in the chunk size asked for, or perhaps, not available just yet - try again later.

In all of these areas, we strive to maintain independence of medium, and retain random trace accessibility without giving up the benefits of disk or network access. We want to make it easy for the user to grab the data and use it anywhere. If possible, we also want to handle segy "similar" formats as though they were segy itself. We also want to permit non-standard segy implementations where there are more than one file per "tape", or where all of the channels do not have the same number of samples. This is very ambitious, and many of these "features" are mutually exclusive.

How:

Like many other software designers, when faced with complexity, we "hide the implementation". The user program refers to a more or less ideal model of what a segy file should be and do, and the interface code makes the underlying available system resources perform the tasks needed as efficiently as possible. "SEGYLIB" uncouples the details of segy access from the system, so these can be simple or complex, and only the SEGYLIB programmer need deal with them, as diagrammed below:



Some assumptions and tradeoffs must be made. We assume that only standard segy is allowed so traces MUST be all one size within a file. Only limited (sequential) multi-file support is allowed on tape. Two binary reel header fields, the number of samples and the data format, MUST be supplied. A third non-standard entry, the number of traces in this file, consistent with Landmark practice, but not generally used, is recommended. We assert that efficiency in translating traces is paramount, but flexibility of translating header entries is more important than speed.

EXAMPLES

Perhaps the best way of introducing SEGYLIB's capabilities is to walk through a simple (but useful) example of a real program we use quite often. It copies from one file to another, especially from disk to tape. Note the parenthesized comments along the right hand side of the page as we go through it (Figure 3).

1) This is the largest number of samples we allow. So long as this number of samples is larger than the ebcidic header and binary header, this is a correct program. Error checks for this potential conflict are not done for simplicity's sake.

2) sgysiz tells you all you need to know about segy on this computer: lengths of headers (ebcdic, binary, and trace) number of bytes in a trace sample what format is native to us? (For format conversions) how many files can we open at one time?

3) Open the input file named "input", readonly (0). If this were a real program instead of a demonstration, we would either ask the user or call a system routine to get the name from the command line. For example, the unix routine "getarg" is used in our version.

4) Open the output file named "output", read/write (1). In either open case, an error in opening it will stop the program.

5) Read, then write, unmodified, the ebcidic trace header. If we wanted, we could translate the ebcdic characters into ascii using subroutine e2a and print them out here.

6) Read the binary header. Once this is read (or written), various useful facts about the particular file become available. These can be retrieved by "sgynbf".

```
c       xsegy.f          a demo program to copy segy files
c
        parameter        ( NSAMP = 10000 )                                  (1)
c
        real     trace(nsamp)
c
c
        call sgysiz( lehdr, lbhdr, lthdr, nbsamp, idhost, maxnf )          (2)
c
        call sgyopn( 1,'input',0,istat )                                   (3)
        if( istat .lt. 0 ) stop 'sgyopn input'
        call sgyopn( 2,'output',1,istat )                                  (4)
        if( istat .lt. 0 ) stop 'sgyopn output'
c
        call sgyrde( 1,trace,istat )                                       (5)
        call sgywte( 2,trace, istat )
c
c paste the number of traces available, if possible
c
        call sgyrdb( 1,trace,istat )                                       (6)
        call sgynbf( 1,nbytes,idtype,ns,ntrace )                           (7)
        call sgyhdp( trace,61,2,ntrace )                                   (8)
        call sgywtb( 2,trace,istat )                                       (9)
c
        if( ns .gt. nsamp-(lthdr/nbsamp) ) then                            (10)
          write(*,*) 'traces too long'
          stop
        endif
        nread = nbytes*ns+lthdr
c
100     call sgyrdt( 1,trace,nread,istat )                                 (11)
        if( istat .ne. nread ) goto 200
          call sgywtt( 2,trace,nread,istat )
          call sgyleft( 2,2350,6250,left )
          if( left .le. 0 ) stop 'oops file is too long!'
          goto 100
c
200     continue
        call sgycls( 1,istat )                                             (12)
        call sgyeof( 2,1,istat )
        call sgycls( 2,istat )
        end
```

fig. 3

7) Find out: number of bytes/sample in this file's format what format is this file in? How many samples/trace in this file? How many traces are in this file? This can be known only when the implementation can determine the size of the file (disk) or when the file was written with the Landmark byte 61 binary header convention.

8) Set the two byte integer (I*2) variable at byte 61 in the binary header to the number of traces in the file. There are 1 and 4 byte integer and ibm floating point flavors too. There is a symmetric "sgyhdg" to import header values. Note that we never need to equivalence arrays or resort to other non-portable coding techniques in order to convert header entries.

9) Write the binary header to the output file.

10) If these traces would be truncated, give up. Again, this is portable, and depends only on SEGYLIB itself.

11) While no special conditions occur, for each input trace, read it, write it. If the output tape would exceed 2350 feet at 6250 bpi, then quit. (This is an example of pragmatism: disk files always have room for one more trace! There are no explicit tape mount/unmount calls in the library at present.)

12) The file has been copied, or some fatal error occurred. Close both files after writing an end of file to the output file.

SUMMARY OF SEGY ROUTINES

Input Routines

Suggested calling order for input:

sgysiz	find out our host data format
sgyopn	open input file
sgynbf	get particulars about this input file number of bytes/trace, number of traces
sgyrdb	(optional) get constants

until eof,

sgyrdt	read traces
sgycvt	convert to internal host format
sgycls	

Output Routines

Suggested calling order for output:

sgysiz	find out our host data format
sgyopn	open output file
sgywte	write ebcdic header set format, number of samples, (optional) number of traces in binary header
sgywtb	write binary header
sgynbf	get particulars about this output file number of bytes/trace, number of traces

until done,

sgycvt	convert from host to output format
sgywtt	write traces
sgyeof	
sgycls	(does not write eofs, since you might not be at end on cls!)

DESCRIPTION OF SEGY ACCESS ROUTINES

GET INFORMATION

File "segypar.fin" can be used to get compile time information except for "maxnf", which is available in "segy.fin". Using "segy.fin" is not recommended.

subroutine sgysiz(lehdr, lbhdr, lthdr, nbsamp, idhost, maxnf)

output:

lehdr	length (bytes) of ebcdic reel header
lbhdr	length (bytes) of binary reel header
lthdr	length (bytes) of prepended trace header
nbsamp	length (bytes) of an internal format segy trace on the calling machine.
idhost	host data type
maxnf	maximum number of simultaneously open files.

subroutine sgynbf(ihandl, nbytes, idtype, nsamp, ntrace)

input:

ihandl	open file handle

output:

nbytes	number of bytes/sample for this file. You should request nsamp*nbytes+LENTHDR if you want to get an entire trace on input.
idtype	format of each sample in this file. This can be used for format conversion.
nsamp	number of samples per trace in this file.
ntrace	number of traces in this file. Only available for disk files and tape files with byte 61 of binary header set. Set to 0 for tape files without such.

These values are available only for open files, and for output files where the binary header has already beenwritten. Otherwise, zeros are returned for each. For output files, ntrace returns the number written so far.

logical function sgytap
sgytap(ihandl)

input:

ihandl	which file number

output:

true if this file is really a tape false if it is not open or is a disk file

FILE OPEN CLOSE

subroutine sgyhdl(ihandl)

output:

ihandl	valid handle number, 1 to MAXNF, or 0 if none free

subroutine sgyopn(ihandl, name, flags, istat)

input:

ihandl	which file number is to be associated with name
name	name of the file or device to open. If the device is /dev/[n]rmt it is treated as a tape drive.
flags	how to open the file: 0 = read only 1 = read and write 2 = read and write, disappear on close. (scratch file)
istat	0 if ok -1 if bad handle or file already open -2 if file could not be opened as requested -3 if file has a format unknown to these routines

subroutine sgycls(ihandl, istat)

input:

ihandl	handle of open file

output:

istat	0 if close ok -1 if close failed

READ DATA

subroutine sgyrde(ihandl, buf, istat)

subroutine sgyrdb(ihandl, buf, istat)

subroutine sgyrdt(ihandl, buf, nr, istat)

input:

ihandl	handle of an open segy file	
buf	array of data to load	
	array must be at least the proper size:	
	e:	3200 bytes
	b:	400 bytes
	t:	240+nbsamp*nsamp
nr	number of bytes to read:	
	+n	truncate read at size of traces
	-n	do no truncation (for headers)
istat	+n	number of bytes actually read
	-n	error

POSITION A FILE

subroutine sgyntr(ihandl, itrace, istat)

input:

ihandl	handle of an open segy file	
itrace	trace number to position to:	
	-1	ebcdic reel header
	-2	binary reel header
	0	return current trace number
	+n	move to n'th trace in the file, 1=first

output:

istat	-n	bad handle or file is not open or headers not yet written on output
	0	positioned to request reel header
	+n	current trace location

WRITE DATA

subroutine sgywte(ihandl, buf, istat)

subroutine sgywtb(ihandl, buf, istat)

subroutine sgywtt(ihandl, buf, nw, istat)

input:

ihandl	handle of an open segy file	
buf	array of data to be written.	
	array must be at least the proper size:	
	e:	3200 bytes
	b:	400 bytes
	t:	240+nbsamp*nsamp
nw	number of bytes to write:	
	+n	truncate write at size of traces
	-n	do no truncation (for headers)

output:

istat	+n	number of bytes actually written
	-n	error

WRITE EOF

call sgyeof(ihandl, n, is)

input:

ihandl	handle of open segy file	
n	number of eofs to write	

output:

istat	=0 if ok
	<0 if an error occurred

FREE SPACE

sgyleft(ihandl, ifeet, idens, left)

input:

ihandl	handle of open segy file
ifeet	number of feet you want to write, maximum
idens	density of recording, bits/inch (1600,6250)

output:

left	number of traces which could be output. <0 file not open =0 no more space; end this tape >0 more space is available. open disk files always return >0.

CONVERT HEADER FORMATS

zero out a header array

subroutine sgyhdz(hdr, nb)

output:

hdr	properly sized header to be set to all zeroes

input:

nb	number of bytes

translate from header to host format

subroutine sgyhdg(hdr, ib, nb, (i,r)val)

input:

hdr	segy header of some kind
ib	byte index, as described in the segy standard
nb	size of the item, 1,2,4 bytes for integer, -4 for ibm float

output:

xval	integer or real value in host format

translate from host to header format

subroutine sgyhdp(hdr, ib, nb, (i,r)val)

output:

hdr	segy header of some kind

input:

ib	byte index, as described in the segy standard
nb	size of the item, 1,2,4 bytes for integer, -4 for ibm float
xval	integer or real value in host format

translate from ebcdic to ascii or ascii to ebcdic

subroutine e2a(hdr,n)

subroutine a2e(hdr,n)

in/output:

hdr	segy header of some kind

input:

n	number of bytes of the header

CONVERT FORMATS

subroutine sgycvt(in, ifmt, ofmt, n, diff, out)

input:

in	array of data as read into memory
ifmt	data type of in
ofmt	data type of out
n	number of samples:
diff	.true. if in and out are different arrays,.false. otherwise

output:

out	array of data to be sent out may be same as in, but make sure to allocate enough space for the type of data you convert!

PPICK PROGRAM IMPLEMENTATION SECTION

design issues
- simplicity
- performance

divisions
- computation
- plotting
- picking
- datafiles

operating system interface
- getarg
- signal
- building from sources and linking with external libraries

summary of files and routines
- use
- walkthroughs

implementation issues and problems

Landmark:
- make'ing fortran programs
- graphics hardware glitches
- network software needs support
- tape support of foreign hardware
- QUIT signal
- closms/wclose interaction
- library ordering under AIX

Masscomp:
- display resolution

Sun:
- cgi use and problems.

Design Issues

Ppick was designed to be a relatively simple interactive display and pick application. This simplicity was desired for two reasons: we wanted to be able to run the application on both Sun and Landmark computers (they have vastly different graphics and mouse support) and we wanted to be able to partition its functionality over a network eventually. These lead to a design which relies as little as possible on the underlying support from the operating system or proprietary libraries. As much functionality as possible is hidden by interface libraries, which will be discussed. "Ppick" was later ported to a Masscomp in an afternoon, by re-writing the mouse and plotting library.

Ppick is an interactive program, so its performance is important, but in its initial implementation, we were less concerned with speed and more with correctness. The vast improvement in overall time to obtain a velocity function via Ppick compared to previous methods leads to its use, even though it is not particularly fast in terms of trace display, and not very elegant in its mouse handling.

logical divisions:

The program is logically divided into four groups of code, each roughly comprising a library or file group. Each group is stored in its own directory descended from "home", as follows:

computation/control
    ppickdir/{*.fin, *.f}
plotting
    sunpltlib.a, sunpltdir/{*.fin, *.f}
picking
    sunpltlib.a, sunpltdir/{*.fin, *.f}
data files
    segylib.a, segydir/{*.c, *.fin, *.f}

Each of these groups will be separately discussed.

computation/control:

This group comprises the main program, and all of the computational routines in the program, which are by and large, straight-forward fortran codes. The main program initializes all files, opens the graphics window and the mouse, draws the initial display, and waits for the user to indicate changes to the existing velocity model.

This group is generally system independent.

plotting:

This group contains a very simple window handler. It knows how to open/close a window, erase it, and draw variable area filled seismic traces to make a panel directly from user time series. It allows horizontal text of one size only, and can draw three kinds of lines: seismic data, timing lines, and trial fit lines. These latter are eraseable by being re-drawn. There are routines also which draw xy lines onto the same arbitrary scale seismic panel as the variable area display, and which provide the mouse routines with the appropriate scale factors so picks can be made in user units. These routines work within windows or on entire screens, depending on the level of support offered by the operating system libraries.

This group is very system dependent, but has been implemented on Sun/unix using SunCGI, Landmark/IBM RT/AIX, and Masscomp/RTU/GP. A version using X11 release 2 or 3 is planned.

picking:

This group contains a very simple mouse handler. It waits until one of 3 buttons is pressed. The coordinates of the mouse are returned in raw window units, in scaled data units, and in nearest trace and sample index form.

This group is very system dependent, but has been implemented on Sun/unix using SunCGI, Landmark/IBM RT/AIX, and Masscomp/RTU/GP. A version using X11 release 2 or 3 is planned.

datafiles:

This group contains a SEG-Y access package which attempts to hide the actual storage details. There are routines to open, close and position segy files on disk or tape. There are routines to get information about the particular file or the system environment, to convert segy reel or trace header data to internal form, and to convert data traces to and from desired forms. Reading, writing and positioning are done so that disk file seems to behave as a tape drive; the "head" is always left in between traces. Since this is done internally without operating system support it is easy to move this package to new systems.

It runs at present on Sun/unix, IBM/AIX, Masscomp/RTU, and Cray/COS. It is modestly system specific.

Operating System Interface

Ppick itself uses only two system routines explicitly. "getarg" is used during program initialization (ppick.f) only to obtain run-time arguments (file names and other options) from the user. "signal" is used only (in tpnmo.f) to allow a lengthy re-display of the window to be skipped by the QUIT signal. Either of these can be removed without significant impact on the overall execution of the program. (In fact, the unix QUIT signal does not seem to be transmitted properly in the Landmark version.)

compiling Ppick:

```
cd <where you loaded Ppick into>
cd segydir
make
cd ../sunpltdir
make
cd ../ppickdir
make
cd ..
ppick test.taup test.mod
```

You should see a panel of traces displayed, with three lines through the zero crossings of the wiggles. Move the mouse to the upper right corner and push button 1, then 2 to exit.

Since Landmark "make" does not in any form support Fortran (even when the required default dependecies are included, the compilation fails in odd ways and the object modules are useless) there is a pseudo-make. In each sub-directory there is an executable shell script which will at least compile and make the expected target. Your environment PATH must reference . before /bin in order to obtain the behavior expected. If this is not the case, type "./make" instead.

libraries:

Landmark: -L/d/lib -llm -llg -lagc \
-llm -llg -lagc \
-luser -lfh -llm \
-lagc -lwt -llm

Masscomp: -lwindow -lgp-fp -lm

Sun: -lcgi77 -lcgi -lsuntool -lsunwindow -lpixrect -lm

These references are embedded in the respective make or Makefile files.

Summary of Files and Routines

Using Ppick:

You type a command line of the form: "ppick args" whereargs is one of the following forms:

taupfile [inmodel [outmodel [ishot [shotspacing [nmofile]]]]]

name	default	
taupfile	'test.taup'	segy tau-p data set; MUST have sample interval, number of samples, delay and source/receiver offset values in trace headers. Must have data type and number of samples/trace in binary reel header. The offset ofthe second trace must be > the first.
inmodel	<none>	text file velocity model; see description of getsp, below.
outmodel	'test.out'	text file velocity model; see description of getsp, below.

These are used for interpolation within a spatially varying velocity model, and can be ignored unless your model has velocities for more than one shot:

ishot	1	initial shot of the input dataset
shotsp	33.33	shot spacing of the input dataset
nmofile	<none>	if not specified, none is created.if specified, the moved out data is saved as a segy file for use by other programs.

Menu buttons:

Landmark mouse button "3" is the keyboard digit "3".

LAYER
- 1: decrement number of selected layer
- 2: increment number of selected layer
- 3: re-draw current model

WINDOW
- 1: pick upper left corner of sub-window
- 2: pick lower right corner of sub-window
- 3: re-draw entire window

RAW
- any: redraw raw tau-p data and current model

NMO-T
- any: redraw nmo in time

NMO-D
- any: redraw nmo in depth

SAVE
- any: append current model to file "test.out"

ADD
- any: make a new layer before existing selected layer; new layer becomes the selected layer.

DEL
- any: delete selected layer

QUIT
- any: exit from program; no SAVE is done. You must confirm QUIT with a a second button.

within data window:
- 1: select p=0 time for selected layer; draw horizontal test line across panel
- 2: type out current parameters for selected layer
- 3: pick new dip and velocity for this layer. v is adjusted by vertical dimension, dip by the distance from the center of the window. The model is redrawn with the new values, but dip is ignored if 1D only is in effect (the default).

below data window:

initial vdelta is 1.0, vmin=0.75, vmax=1.75
- 1: decrease velocity window by vdelta
- 2: increase velocity window by vdelta
- 3: change vdelta: 0.5, 1.0, 1.5 rollover vmax = vmin + vdelta

Descriptions of Routines

segylib.a seg-y file support library

Read the general writeup in the beginning of file "segy.f".

sunpltlib.a plotting/picking support library

Read the general use notes in the beginning of file "wopen.f", "mtsplt.f" and "mouse.f".

wopen.f:

wopen:	open a window
wclear:	clear a window
wclose:	close a window

mtsplt.f:

mtsplt:	plots a panel of seismic traces in a window
xyline:	plots a line in the same coordinate space as mtsplt
mtsnew:	reset screen pointers to redraw panel
mtsmap:	publish screen to data scale factors

lintyp.f:

set line bundle characteristics

ptext.f:

output text string

ivafill.f:

does general variable area filled x,y plotting

mouse.f:

initms:	open the mouse
getms:	wait until a button is pushed, return coordinates
closms:	close the mouse

Description of Routines:

segypar.fin

defines byte indicies and header sizes for standard SEG-Y files.
SHOT number is not standardized usage, but is unused by Ppick.

ppick.f:

The program determines the available size of the current window, uses system routine "getarg" to read the argument list, and opens the input file. It loads the user specified or default values into all of the other parameters, and reads the input model (if any). If no input model exists, Ppick creates one thick layer by default. It opens the nmo output file (if specified), and begins to read the input tau-p file. It reads the first trace to determine the sample interval, time of the first sample and the ray parameter (source receiver offset). It determines the fixed delta ray-parameter from the difference in ray-parameter of these two traces; it is an undetected error if the delta does not remain constant for the entire file. The ray-parameters MUST increase within the file.

The graphics window is opened and the mouse initialized. The default display window is set so the entire time and ray extent is visible. If there are more than 50,000 samples in the panel, then every 20th trace is displayed instead of each trace to speed the initial gross display. Routine "restore" is called to redisplay the entire current window.

There are two velocity models within the program: the reference model which was last used to draw the normal moved out data, and the current trial model which the user is testing. Initially (and always for raw un-moved out tau-p data) they are set the same.

The program waits for a mouse action from the user, as described in the previous section on usage. Logical function "pickit" hides how the menu choices are actually done; the current implementation is a very simple point at a word near the top of the window. There are three areas in the window: the upper row of menu pick items, the middle area which determines velocity or time, and the lower area which sets the limits and resolution of the velocity area. If necessary, the old model is erased and the current model is drawn over the seismic data. Picking in the middle area (velocity changes) can only cause the redrawing of these model lines. The coordinates come from combining "pvtauXd" (for the trial model), and "nmotpXd" (for the reference model) in the normal moveout case, or just "pvtauXd" for the raw tau-p case. Some menu items may require a full redraw of the window. For example, if a new time or ray-parameter window is chosen, Ppick calls "restore" to perform this. The mouse loop is terminated only by an interrupt or choosing the QUIT button twice. All files are closed, and the display state (if it was saved) is restored before the program exits.

logical function pickit
returns true if the current location of the mouse is close enough to a menu button.

ppick.fin
defines sizes of arrays, where the menu buttons are, and establishescommon areas for use by all other routines.

getsp.f
handles velocity function bookkeeping

getsp:
loads a velocity model from a text file, and does interpolation to determine the velocity model for current shot location.format of model file:number of shot locations in this model file <integer> repeat for number of shot locations: this shot<integer>, number of layers here <integer>repeat for number of layers:velocity, thickness, dip, density <reals> for example (this is file test.mod):

```
1
1       4
1.8     0.75    0.0     1.0
3.6     0.5     0.0     1.0
1.8     0.5     0.0     1.0
5.0     1.0     0.0     1.0
```

layadd:
adds a layer above the current layer if there is room in the internal tables, and slides those below downward.

laydel:
removes current layer (if not the only layer) and slides the rest upwards.

putsp:
outputs the current trial model as a "getsp" readable file.

tr2ref:
copies the current trial model to the reference model.

ref2tr:
copies the reference model to the trial model.

restore.f

This routine handles all re-display (or initial display) of a panel of seismic traces. If a re-display is not currently in progress, the window is cleared, the menu buttons are written at the top of the window, the panel of seismic traces is drawn (tpnmo), and the line style set to draw the model lines. Called by ppick.f to draw the screen initially or after significant change. For the Sun version, it is also called automagically (from cgi) whenever the window is covered or uncovered, or changes size. tpnmo.f This routine is the meat of the window display. It is called when the entire window must be updated for any reason. It determines the current window size, sets a QUIT signal trap handler, and then reads the open input tau-p data file:

For each trace in the input file which should be visible: if this trace is to be skipped (for speed), then position one trace over.

else

read the segy trace, convert it, remove average bias in the window (dcbias). if normal moveout wanted, do it (tpnmoXd) and update the stack/semblance. Also save the moved out trace to the nmo file if wanted.plot the raw or moved out trace.

end

if normal moveout wanted, then leave some blank space, plot the stack/semblance trace.

end

draw the velocity model lines, using pvtauXd (for trial model) and nmotpXd (for reference model) for normal moveout case, or just pvtauXd for raw tau-p case unlink the QUIT signal handler done

Called by "restore.f".

pvtau1d.f

This routine calls tau2d.f for a suite of ray parameters and collects the tau times for the current model.

nmotp1d.f

Builds the nmo corrected tau-p delay time versus ray parameter arrays needed for each layer. We input the computed delay times for the ray parameters in p for a given (reference) mode. These represent the tau-p trajectories for this mode. We use these times and the current model described by z, vel, dip to predict the nmo corrections to the original tau's but for the current mode. The p's correspond, but the new tau's will be returned.

stackem.f

The input is the sum of the nmo corrected data seismograms for several ray parameters and optionally the sum of the energy of these seismograms. The number os samples at each time in the stack are used to normalize the stack. Semblance can also be computed or the stack multiplied by semblance, i.e., a coherency weighted stack.

tau2d.f

For a given input velocity thickness and dip model and for the observed surface ray at the receiver tau2d.f traces the ray for all the model layers and returns: total vertical delay times and the complex acoustic reflection coefficients.

tpnmo1d.f

For the observed surface ray at the receiver compute the 1d tau-p nmo correction times and apply them to the observed seismic ray parameter trace. An output trace is generated by

tracing the ray through all required layers and computing the nmo corrections. The input data trace is sampled via linear interpolation to generate the nmo corrected output trace.

accoef.f

Compute the plane wave complex acoustic reflection coefficient for the current interface for the specified interface reflection angle. The velocity and density of the overlying and underlying layers are used to generate the complex reflection coefficient.

csth.f

A complex function that computes the term sqrt (1.-p**2v**2) where p is the ray parameter and v is the velocity. the result is either real or imaginary depending on the sign of the square root argument.

rads.f

Makes radians from degrees.

isflat.f

Logical function which returns true if all layers in the current model are sufficiently horizontal, < 0.1 degree.
Called by ppick.f and tpnmo.f if in 2-d mode to revert to faster 1-d case if true.

debias.f

Removes the average (within the display window) from each input trace before moveout or display. Called by tpnmo.f.

pvtau2d.f
nmotp2d.f
tpnmo2d.f

These routines are stubs for a yet to be implemented 2d extension to the existing 1d code. If called, they will cause program termination.

Implementation issues:

Landmark:

Make does not work for Fortran programs. There are no default rules for .f.o, and if these are taken from the Sun (which otherwise is very similar) the resulting object files are unusable.

Graphics hardware has many glitches in it. The program bugsdir/?.f demonstrates some of them. The most pertinent are lack of fill when triangles are very acute, or filling OUTSIDE of a polygon. The graphics hardware will always hang the system eventually if you draw enough filled polygons (5-15 minutes). Overlay planes don't show white color. Egaoff/on is a nuisance during development. Control-z does NOT work during Ppick, and hangs the system.

The micom/interlan tcp/ip software was delivered with no install guide, and it constantly transmits arp/icmp packets to nearly ALL hosts on our campus, declaring the Landmark "unreachable". We use our network bridge to filter this traffic, as our campus network administators get testy when we flood their network with garbage, but this also means that the Landmark can't access hosts outside UTIG directly (most notably the Cray). The interface is also slow, and rcpy hangs for large files. If lots of output is sent to an rlogin session in a big hurry (cat bigfile), it usually hangs.

There are no .h files in /usr/include to allow ioctl access to the scsi 1/2 inch tape drives.

The QUIT signal does not seem to be handled properly in Ppick. It is always ignored. A small test program works however, and the same source code works fine on both Sun and Masscomp.

"closms" must be called before "wclose" due to (unfortunate) internal dependencies in sunpltlib.a. "Wopen" must be called before "initms", but this seems natural anyway.

Order of libraries under an AIX ld is touchy and can lead to very difficult to debug load errors. Landmark documentation does not give the complete Landmark search order. Masscomp: 800x600 display resolution is not enough, although windowing gets around the problem somewhat.

Sun:

If Suntools is not running, sunpltlib takes over the hardware window, and colors are then inverted.

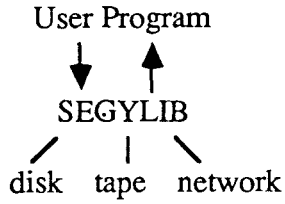
Cgi is slow, and has positioning bugs when retained windows are used.

Color display does not work?

SEG-Y FILE ACCESS PACKAGE - PROGRAM DOCUMENTATION

How:

Like many other software designers, when faced with complexity, we "hide the implementation". The user program refers to a more or less ideal model of what a segy file should be and do, and the interface code makes the underlying available system resources perform the tasks needed as efficiently as possible. "SEGYLIB" uncouples the details of segy access from the system, so these can be simple or complex, and only the SEGYLIB programmer need deal with them, as diagrammed below:



Some assumptions and tradeoffs must be made. We assume that only standard segy is allowed so traces MUST be all one size within a file. Only limited (sequential) multi-file support is allowed on tape. Two binary reel header fields, the number of samples and the data format, MUST be supplied. A third non-standard entry, the number of traces in this file, consistent with Landmark practice, but not generally used, is recommended. We assert that efficiency in translating traces is paramount, but flexibility of translating header entries is more important than speed.

These routines access SEG-Y files on disk or tape with the following properties:

A SEG-Y (hereafter segy) file is comprised of:

A 3200 byte reel header, consisting of 40 80 byte card images. Each byte is an EBCDIC character, and must be converted to the internal character set before program use. A 400 byte reel header, consisting of 60 bytes of both integer*4 and integer*2 data, and a 340 byte area of optional use not defined by the standard. These routines REQUIRE that the

number of samples in bytes 21-22 and the format code in bytes 25-26 be present. If the file is a tape file, the number of traces in the file may be coded in bytes 61,62.

Zero or more data traces, each comprised of:

A 240 byte trace header, consisting of integer*4 and integer*2 data in the first 180 bytes, and 60 bytes of optional information not defined by the standard.

A time series, defined by the data format code and the number of samples. All data traces must be the same length according to the standard. The size of a sample MAY depend according to the system the data was generated on. These routines will understand non-standard segy data format types, some of which may not be the standard 2 or 4 byte lengths. an end of file flag: which is usually implicit, not physically existing, except on tape.

A disk file can be loaded from tape by a command similar to:

```
dd if=/dev/rmt0 bs=32000 of=diskfile
```

This format is the same as Landmark BCM .SEGYOUT primitive generates. It is not the same as Landmark Desktop Utility TODVXS generates, nor is it the same as CGG segyout on disk. All SEGY on tape are the same, however.

For best compatibility between tape and disk, observe a few rules:

Reading less than a whole tape record on the sun gives an error, but gets the correct data. Always ask for the same or more than you believe is there. On disk files, asking for less is not a problem.

Backspacing tape is very slow on the Sun. Don't do it if you can avoid it.

Tape devices can't randomly write without destroying data which already exists beyond the writing point. This works on disk. Don't do it unless you must.

There is no way of knowing how many traces there will be on an input tape. Use the Landmark convention of number of traces in byte 61, 62 of the binary header when you create the tape if you require this capability. If this is not practical, use dd to load the tape to a disk file before use.

SUMMARY OF SEGY ROUTINES

Input Routines

Suggested calling order for input:

sgysiz	find out our host data format
sgyopn	open input file
sgynbf	get particulars about this input file number of bytes/trace, number of traces
sgyrdb	(optional) get constants

until eof,

sgyrdt	read traces
sgycvt	convert to internal host format
sgycls	

Output Routines

Suggested calling order for output:

sgysiz	find out our host data format
sgyopn	open output file
sgywte	write ebcdic header set format, number of samples, (optional) number of traces in binary header
sgywtb	write binary header
sgynbf	get particulars about this output file number of bytes/trace, number of traces

until done,

sgycvt	convert from host to output format
sgywtt	write traces
sgyeof	
sgycls	(does not write eofs, since you might not be at end on cls!)

DESCRIPTION OF SEGY ACCESS ROUTINES

GET INFORMATION

File "segypar.fin" can be used to get compile time information except for "maxnf", which is available in "segy.fin". Using "segy.fin" is not recommended.

subroutine sgysiz(lehdr, lbhdr, lthdr, nbsamp, idhost, maxnf)

output:

lehdr	length (bytes) of ebcdic reel header
lbhdr	length (bytes) of binary reel header
lthdr	length (bytes) of prepended trace header
nbsamp	length (bytes) of an internal format segy trace on the calling machine.
idhost	host data type
maxnf	maximum number of simultaneously open files.

subroutine sgynbf(ihandl, nbytes, idtype, nsamp, ntrace)

input:

ihandl	open file handle

output:

nbytes	number of bytes/sample for this file. You should request nsamp*nbytes+LENTHDR if you want to get an entire trace on input.
idtype	format of each sample in this file. This can be used for format conversion.
nsamp	number of samples per trace in this file.
ntrace	number of traces in this file. Only available for disk files and tape files with byte 61 of binary header set. Set to 0 for tape files without such.

These values are available only for open files, and for output files where the binary header has already beenwritten. Otherwise, zeros are returned for each. For output files, ntrace returns the number written so far.

logical function sgytap
sgytap(ihandl)

input:

ihandl	which file number

output:

true if this file is really a tape false if it is not open or is a disk file

FILE OPEN CLOSE

subroutine sgyhdl(ihandl)

output:

ihandl	valid handle number, 1 to MAXNF, or 0 if none free

subroutine sgyopn(ihandl, name, flags, istat)

input:

ihandl	which file number is to be associated with name
name	name of the file or device to open. If the device is /dev/[n]rmt it is treated as a tape drive.
flags	how to open the file: 0 = read only 1 = read and write 2 = read and write, disappear on close. (scratch file)
istat	0 if ok -1 if bad handle or file already open -2 if file could not be opened as requested -3 if file has a format unknown to these routines

subroutine sgycls(ihandl, istat)

input:

ihandl	handle of open file

output:

istat	0 if close ok -1 if close failed

READ DATA

subroutine sgyrde(ihandl, buf, istat)

subroutine sgyrdb(ihandl, buf, istat)

subroutine sgyrdt(ihandl, buf, nr, istat)

input:

ihandl	handle of an open segy file	
buf	array of data to load	
	array must be at least the proper size:	
	e:	3200 bytes
	b:	400 bytes
	t:	240+nbsamp*nsamp
nr	number of bytes to read:	
	+n	truncate read at size of traces
	-n	do no truncation (for headers)
istat	+n	number of bytes actually read
	-n	error

POSITION A FILE

subroutine sgyntr(ihandl, itrace, istat)

input:

ihandl	handle of an open segy file	
itrace	trace number to position to:	
	-1	ebcdic reel header
	-2	binary reel header
	0	return current trace number
	+n	move to n'th trace in the file, 1=first

output:

istat	-n	bad handle or file is not open or headers not yet written on output
	0	positioned to request reel header
	+n	current trace location

WRITE DATA

subroutine sgywte(ihandl, buf, istat)

subroutine sgywtb(ihandl, buf, istat)

subroutine sgywtt(ihandl, buf, nw, istat)

input:

ihandl	handle of an open segy file	
buf	array of data to be written.	
	array must be at least the proper size:	
	e:	3200 bytes
	b:	400 bytes
	t:	240+nbsamp*nsamp
nw	number of bytes to write:	
	+n	truncate write at size of traces
	-n	do no truncation (for headers)

output:

istat	+n	number of bytes actually written
	-n	error

WRITE EOF

call sgyeof(ihandl, n, is)

input:

ihandl	handle of open segy file
n	number of eofs to write

output:

istat	=0 if ok
	<0 if an error occurred

FREE SPACE

sgyleft(ihandl, ifeet, idens, left)

input:

ihandl	handle of open segy file
ifeet	number of feet you want to write, maximum
idens	density of recording, bits/inch (1600,6250)

output:

left	number of traces which could be output.
	<0 file not open
	=0 no more space; end this tape
	>0 more space is available.
	open disk files always return >0.

CONVERT HEADER FORMATS

zero out a header array

subroutine sgyhdz(hdr, nb)

output:

hdr	properly sized header to be set to all zeroes

input:

nb	number of bytes

translate from header to host format

subroutine sgyhdg(hdr, ib, nb, (i,r)val)

input:

hdr	segy header of some kind
ib	byte index, as described in the segy standard
nb	size of the item, 1,2,4 bytes for integer, -4 for ibm float

output:

xval	integer or real value in host format

translate from host to header format

subroutine sgyhdp(hdr, ib, nb, (i,r)val)

output:

hdr	segy header of some kind

input:

ib	byte index, as described in the segy standard
nb	size of the item, 1,2,4 bytes for integer, -4 for ibm float
xval	integer or real value in host format

translate from ebcdic to ascii or ascii to ebcdic

subroutine e2a(hdr,n)

subroutine a2e(hdr,n)

in/output:

hdr	segy header of some kind

input:

n	number of bytes of the header

CONVERT FORMATS

subroutine sgycvt(in, ifmt, ofmt, n, diff, out)

input:

in	array of data as read into memory
ifmt	data type of in
ofmt	data type of out
n	number of samples:
diff	.true. if in and out are different arrays,.false. otherwise

output:

out	array of data to be sent out may be same as in, but make sure to allocate enough space for the type of data you convert!

EXAMPLES

Perhaps the best way of introducing SEGYLIB's capabilities is to walk through a simple (but useful) example of a real program we use quite often. It copies from one file to another, especially from disk to tape. Note the parenthesized comments along the right hand side of the page as we go through it (Figure 1).

1) This is the largest number of samples we allow. So long as this number of samples is larger than the ebcidic header and binary header, this is a correct program. Error checks for this potential conflict are not done for simplicity's sake.

2) sgysiz tells you all you need to know about segy on this computer: lengths of headers (ebcdic, binary, and trace) number of bytes in a trace sample what format is native to us? (For format conversions) how many files can we open at one time?

3) Open the input file named "input", readonly (0). If this were a real program instead of a demonstration, we would either ask the user or call a system routine to get the name from the command line. For example, the unix routine "getarg" is used in our version.

4) Open the output file named "output", read/write (1). In either open case, an error in opening it will stop the program.

5) Read, then write, unmodified, the ebcidic trace header. If we wanted, we could translate the ebcdic characters into ascii using subroutine e2a and print them out here.

6) Read the binary header. Once this is read (or written), various useful facts about the particular file become available. These can be retrieved by "sgynbf".

```
c       xsegy.f          a demo program to copy segy files
c
        parameter        ( NSAMP = 10000 )                              (1)
c
        real    trace(nsamp)
c
c
        call sgysiz( lehdr, lbhdr, lthdr, nbsamp, idhost, maxnf )      (2)
c
        call sgyopn( 1,'input',0,istat )                               (3)
        if( istat .lt. 0 ) stop 'sgyopn input'
        call sgyopn( 2,'output',1,istat )                              (4)
        if( istat .lt. 0 ) stop 'sgyopn output'
c
        call sgyrde( 1,trace,istat )                                   (5)
        call sgywte( 2,trace, istat )
c
c paste the number of traces available, if possible
c
        call sgyrdb( 1,trace,istat )                                   (6)
        call sgynbf( 1,nbytes,idtype,ns,ntrace )                       (7)
        call sgyhdp( trace,61,2,ntrace )                               (8)
        call sgywtb( 2,trace,istat )                                   (9)
c
        if( ns .gt. nsamp-(lthdr/nbsamp) ) then                        (10)
          write(*,*) 'traces too long'
          stop
        endif
        nread = nbytes*ns+lthdr
c
100     call sgyrdt( 1,trace,nread,istat )                             (11)
        if( istat .ne. nread ) goto 200
          call sgywtt( 2,trace,nread,istat )
          call sgyleft( 2,2350,6250,left )
          if( left .le. 0 ) stop 'oops file is too long!'
          goto 100
c
200     continue
        call sgycls( 1,istat )                                         (12)
        call sgyeof( 2,1,istat )
        call sgycls( 2,istat )
        end
```

fig. 1

7) Find out: number of bytes/sample in this file's format what format is this file in? How many samples/trace in this file? How many traces are in this file? This can be known only when the implementation can determine the size of the file (disk) or when the file was written with the Landmark byte 61 binary header convention.

8) Set the two byte integer (I*2) variable at byte 61 in the binary header to the number of traces in the file. There are 1 and 4 byte integer and ibm floating point flavors too. There is a symmetric "sgyhdg" to import header values. Note that we never need to equivalence arrays or resort to other non-portable coding techniques in order to convert header entries.

9) Write the binary header to the output file.

10) If these traces would be truncated, give up. Again, this is portable, and depends only on SEGYLIB itself.

11) While no special conditions occur, for each input trace, read it, write it. If the output tape would exceed 2350 feet at 6250 bpi, then quit. (This is an example of pragmatism: disk files always have room for one more trace! There are no explicit tape mount/unmount calls in the library at present.)

12) The file has been copied, or some fatal error occurred. Close both files after writing an end of file to the output file.